# The Transformer architecture

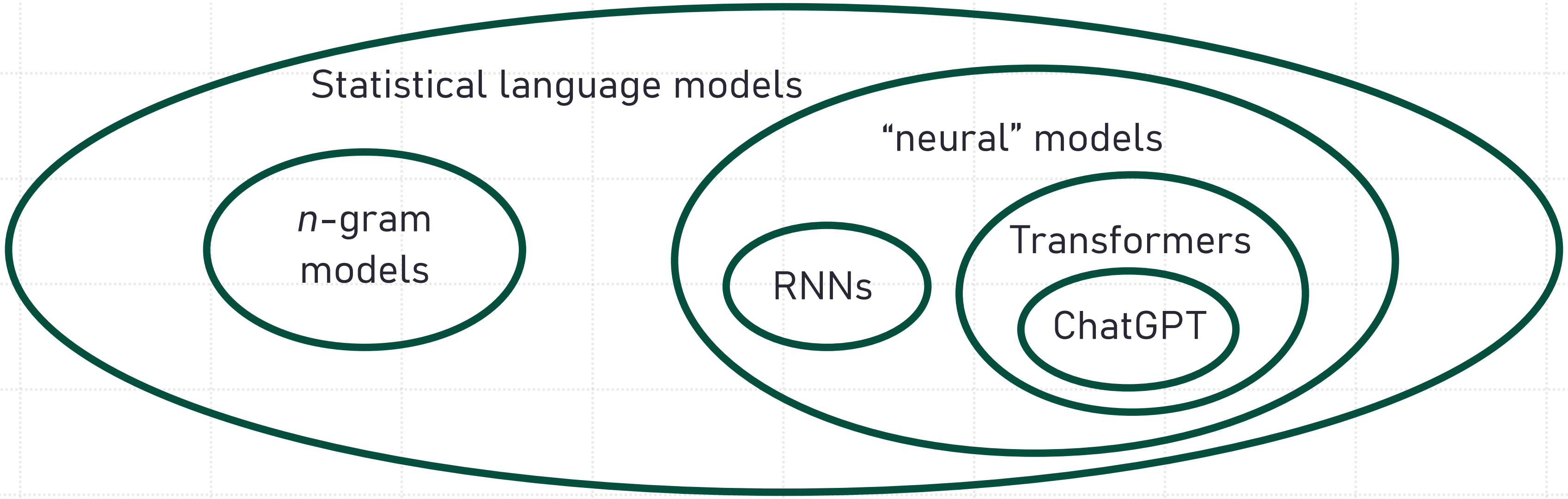
*COGS 150*

*Sean Trott*

*Winter 2024*

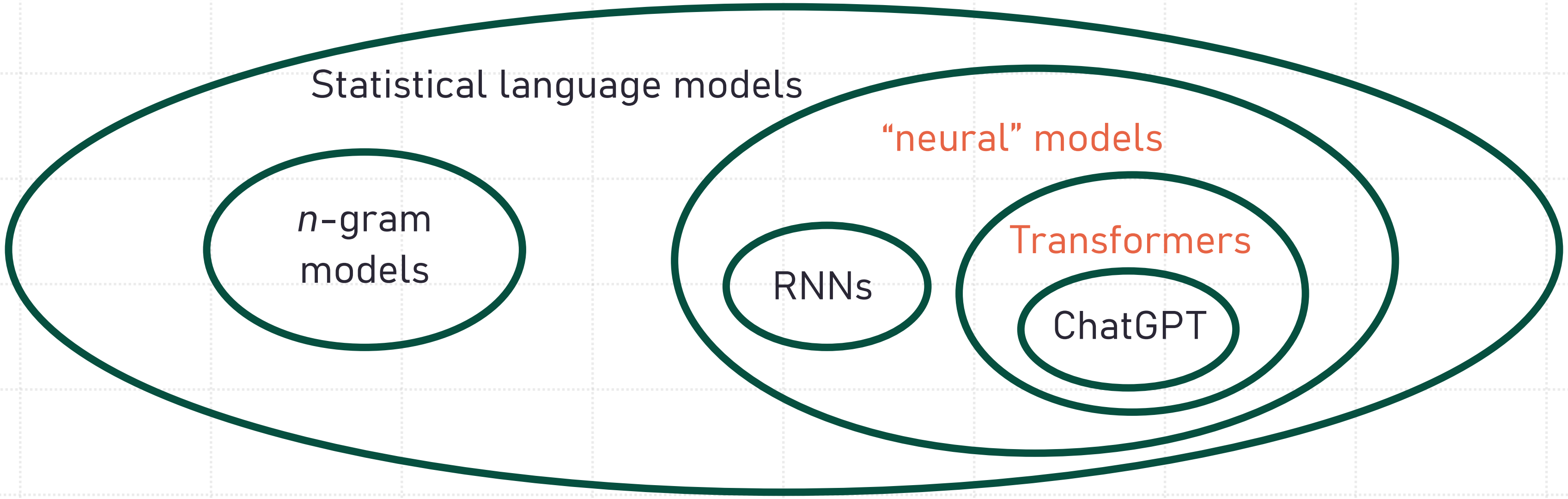
# A brief taxonomy

- Many approaches to **language modeling**.
- All revolve around **statistical learning** in some way.



# A brief taxonomy

- Many approaches to **language modeling**.
- All revolve around **statistical learning** in some way.

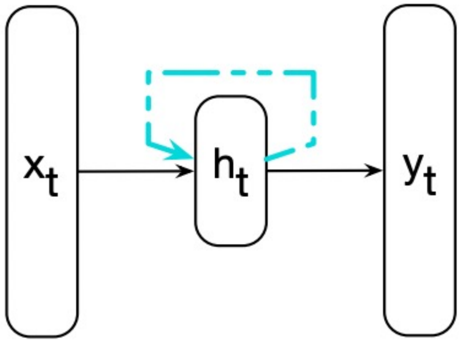


# Lecture plan

- “Attention”: high-level introduction and motivation.
- The transformer architecture—is attention all you need?
- The advent of “pre-trained LLMs”.

# Lecture plan

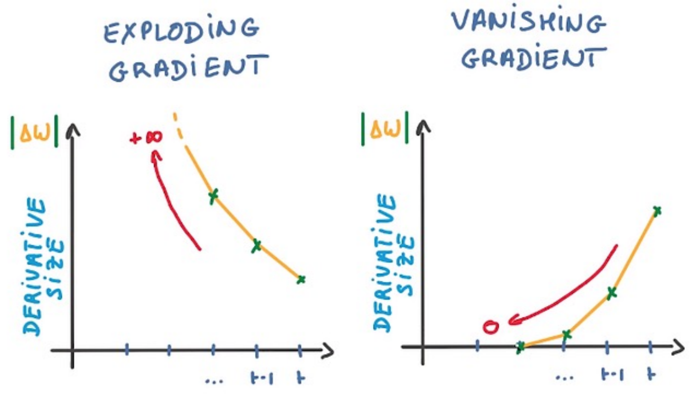
- “Attention”: high-level introduction and motivation.
- The transformer architecture—is attention all you need?
- The advent of “pre-trained LLMs”.




# RNNs: recap, and limitations

- A **recurrent neural network (RNN)** has at least one *recurrent* connection, which acts as a kind of "memory" of the context.
- RNNs work pretty well, but do have limitations.

**Limitation #1**:
Vanishing/exploding gradient.




**Limitation #2**:
Training is hard to parallelize.

Recurrent structure makes it hard to process many batches in parallel—harder to take advantage of compute.

# The advent of “attention”

**Attention** is a mechanism that—metaphorically—allows an LLM to “focus” (or “attend”) on specific elements in a sequence.

- Often, <u>accurate predictions</u> depend on words from a while ago.

Check the program log and find out whether it ran please.

Check the battery log and find out whether it ran down please.

# The advent of "attention"

**Attention** is a mechanism that—metaphorically—allows an LLM to "focus" (or "attend") on specific elements in a sequence.

- Often, accurate predictions depend on words from a while ago.

Check the **program** log and find out whether it **ran please**.

Check the **battery** log and find out whether it **ran down** please.

*…whether it ran ____*

- Knowing what comes next depends on looking far back in the sequence.

# The advent of “attention”

**Attention** is a mechanism that—metaphorically—allows an LLM to “focus” (or “attend”) on specific elements in a sequence.

- Often, accurate predictions depend on words from a while ago.

Check the **program** log and find out whether it **ran please**.

Check the **battery** log and find out whether it **ran down** please.

*…whether it ran ____*

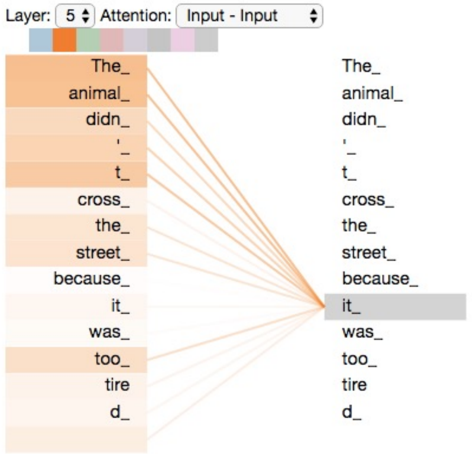
- Knowing what comes next depends on looking far back in the sequence.

# The advent of “attention”

**Attention** is a mechanism that—metaphorically—allows an LLM to “focus” (or “attend”) on specific elements in a sequence.

- Often, accurate predictions depend on words from a while ago.
- This also helps identify relationships between elements in the sequence.

The animal didn’t cross the street because it was tired.

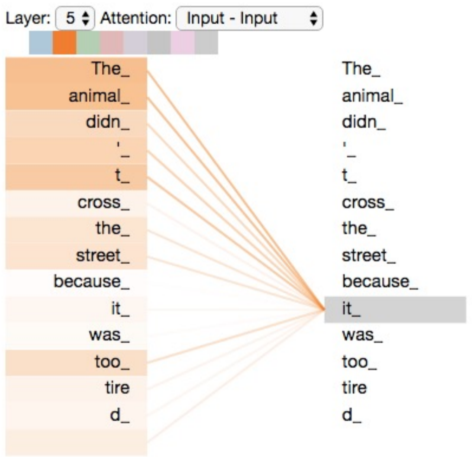


# The advent of “attention”

**Attention** is a mechanism that—metaphorically—allows an LLM to “focus” (or “attend”) on specific elements in a sequence.

- Often, <u>accurate predictions</u> depend on words from a while ago.
- This also helps identify <u>relationships</u> between elements in the sequence.

The **animal** didn’t cross the street because **it** was tired.



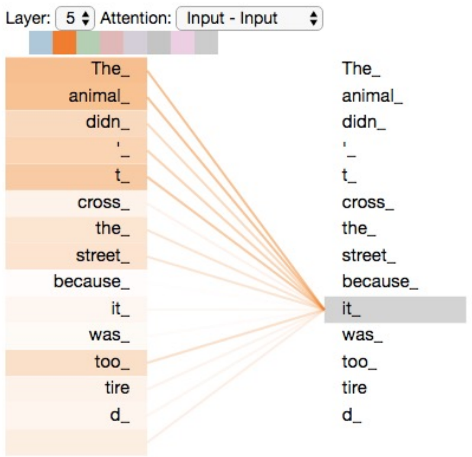
But how does this actually <u>work</u>?

# The advent of “attention”

**Attention** is a mechanism that—metaphorically—allows an LLM to “focus” (or “attend”) on specific elements in a sequence.

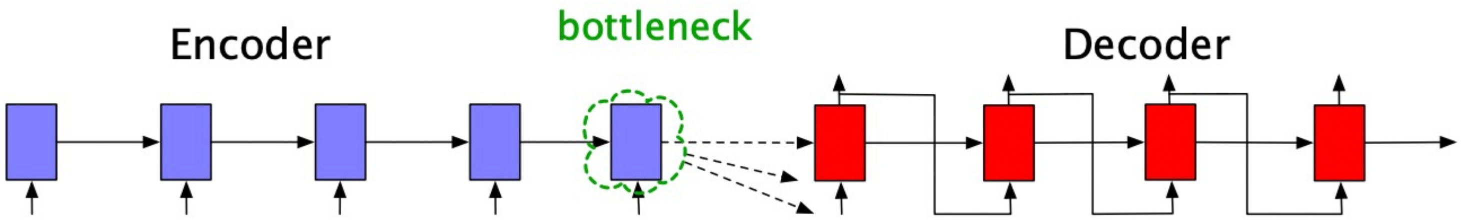
- Often, <u>accurate predictions</u> depend on words from a while ago.
- This also helps identify <u>relationships</u> between elements in the sequence.

The **animal** didn’t cross the street because **it** was tired.



# Attention: the origins

- Originally, attention was developed to help with **machine translation**.
- Traditional, RNN-based translation models had a “bottleneck” in their design.

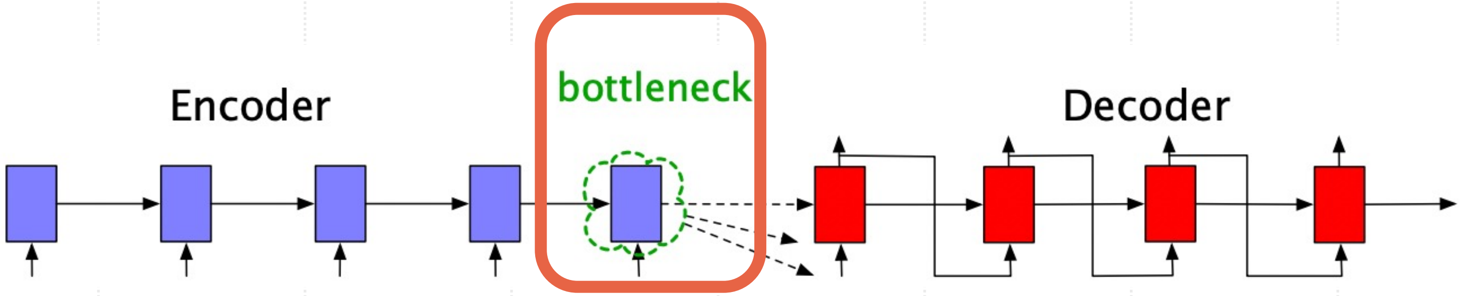



Encodes source language.

Decodes to target language.

# Attention: the origins

- Originally, attention was developed to help with **machine translation**.
- Traditional, RNN-based translation models had a “bottleneck” in their design.




Encodes source language.

Decodes to target language.

**Bottleneck**: *all* the information required to translate a sentence must be packed into this last hidden state.

# Attention: the origins

- Originally, attention was developed to help with **machine translation**.
- Traditional, RNN-based translation models had a “bottleneck” in their design.
- Attention is a mechanism for putting *all* those hidden states into a single fixed-length vector—by focusing on what’s most relevant.

**Dot-product attention**: implements “relevance” as *embedding similarity.*

To illustrate this, let’s look at an example from a domain we’re already familiar with—language modeling.

# Dot-product attention: illustrated

In **dot-product attention**, the dot product between every pair of words is used to build a custom, context-dependent vector.

The cat sat on ___

"on"

The

cat

sat

on

# Dot-product attention: illustrated

In **dot-product attention**, the dot product between every pair of words is used to build a custom, context-dependent vector.

The cat sat on ___

“on”

| | |
|---|---|
| V1 | The |
| V2 | cat |
| V3 | sat |
| V4 | on |

# Dot-product attention: illustrated

In **dot-product attention**, the dot product between every pair of words is used to build a custom, context-dependent vector.

The cat sat on ___

“on”

| | |
|---|---|
| V1 | The |
| V2 | cat |
| V3 | sat |
| V4 | on |

Each of these is represented by an **embedding**.

The dot product captures the **similarity** in embeddings.

# Dot-product attention: illustrated

In **dot-product attention**, the dot product between every pair of words is used to build a custom, context-dependent vector.

The cat sat on ___

| | | "on" | $w * c$ |
|---|---|---|---|
| V1 | The | | .2 |
| V2 | cat | | .1 |
| V3 | sat | | .1 |
| V4 | on | | 1 |

Numbers made up for illustration purposes!

# Dot-product attention: illustrated

In **dot-product attention**, the dot product between every pair of words is used to build a custom, context-dependent vector.

The cat sat on ___

| | | "on" | $w * c$ | $\sigma(x)_j = \frac{e^{x_j}}{\sum_k e^{x_k}}$ |
|---|---|---|---|---|
| V1 | The | | .2 | .2 |
| V2 | cat | | .1 | .18 |
| V3 | sat | | .1 | .18 |
| V4 | on | | 1 | .44 |

Now, we **soft-max** these values to create a probability distribution.

# Dot-product attention: illustrated

In **dot-product attention**, the dot product between every pair of words is used to build a custom, context-dependent vector.

The cat sat on ___

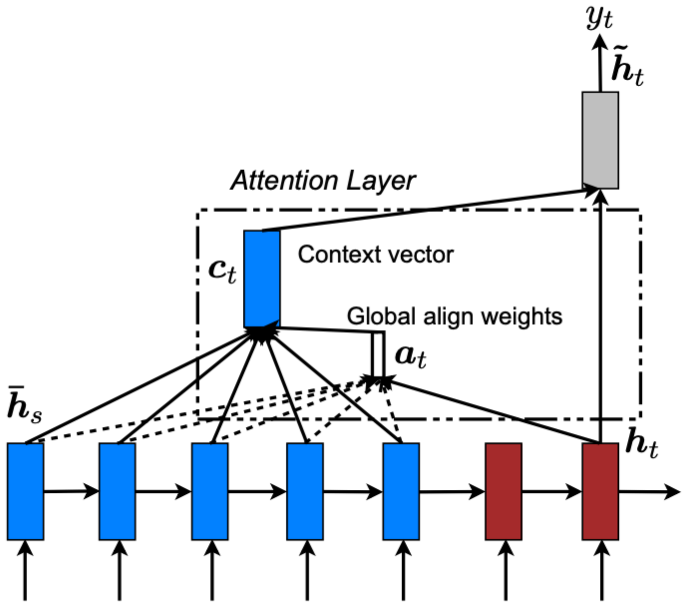
| | "on" | $w * c$ | $\sigma(x)_j = \frac{e^{x_j}}{\sum_k e^{x_k}}$ |
|---|---|---|---|
| V1 | The | .2 | .2 |
| V2 | cat | .1 | .18 |
| V3 | sat | .1 | .18 |
| V4 | on | 1 | .44 |

These are our **attention weights**.

Each represents the "relevance" of $V_n$ to "on".

# Dot-product attention: illustrated

In **dot-product attention**, the dot product between every pair of words is used to build a custom, context-dependent vector.



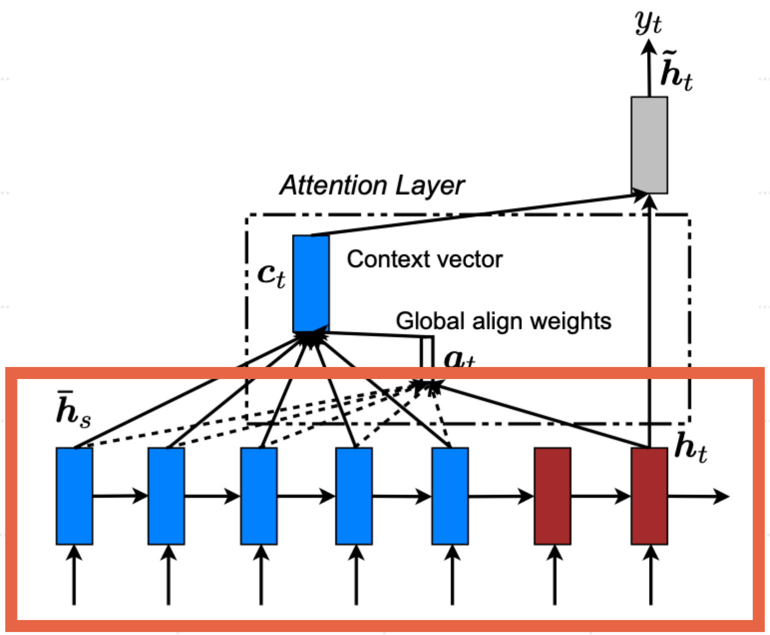

Now, compute **weighted average** over all hidden states—using these attention scores as "weights"!

$$\mathbf{c}_i = \sum_j \alpha_{ij}\, \mathbf{h}^e_j$$

# Dot-product attention: illustrated

In **dot-product attention**, the dot product between every pair of words is used to build a custom, context-dependent vector.

Hidden states



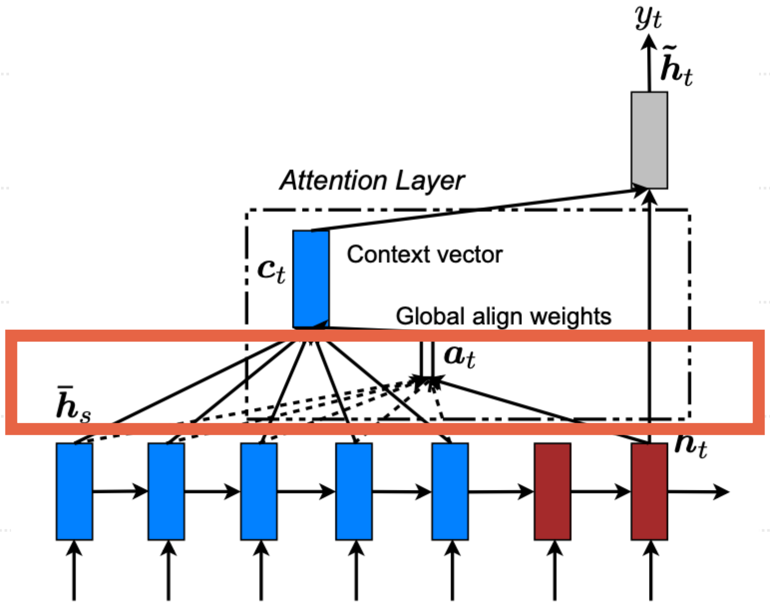

Now, compute **weighted average** over all hidden states—using these attention scores as "weights"!

$$\mathbf{c}_i = \sum_j \alpha_{ij} \mathbf{h}_j^e$$

# Dot-product attention: illustrated

In **dot-product attention**, the dot product between every pair of words is used to build a custom, context-dependent vector.

Compute attention weights



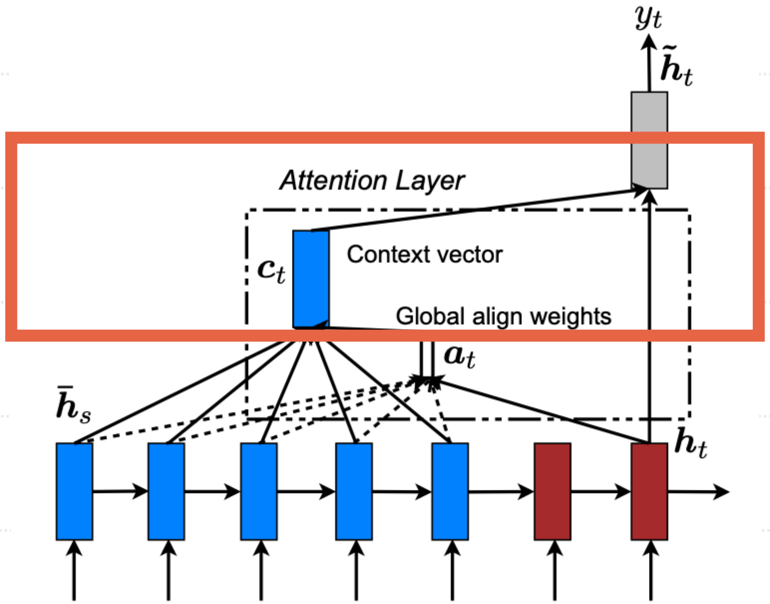

Now, compute **weighted average** over all hidden states—using these attention scores as "weights"!

$$\mathbf{c}_i = \sum_j \alpha_{ij}\, \mathbf{h}_j^e$$

# Dot-product attention: illustrated

In **dot-product attention**, the dot product between every pair of words is used to build a custom, context-dependent vector.

Use attention weights to create new **context vector**.



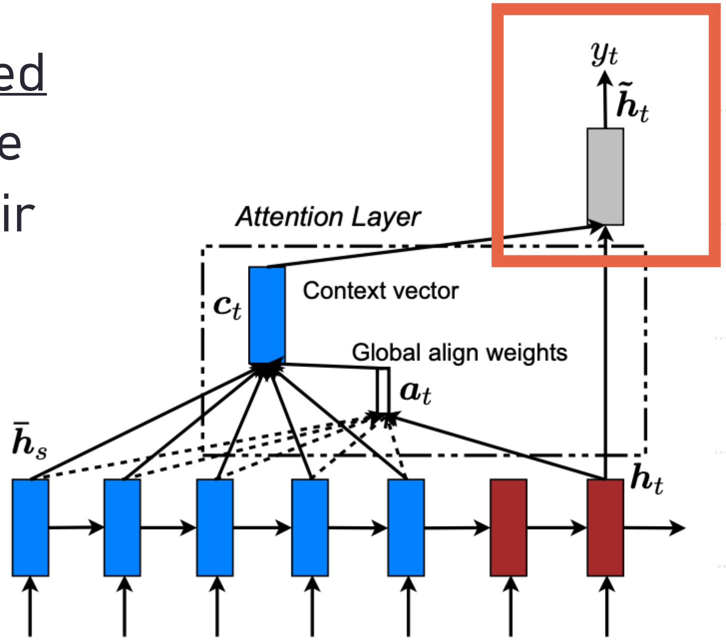

Now, compute **weighted average** over all hidden states—using these attention scores as "weights"!

$$\mathbf{c}_i = \sum_j \alpha_{ij} \mathbf{h}_j^e$$

# Dot-product attention: illustrated

In **dot-product attention**, the dot product between every pair of words is used to build a custom, context-dependent vector.

Predictions are now <u>weighted</u> by different elements of the sequence depending on their "relevance".




Now, compute **weighted average** over all hidden states—using these <u>attention scores</u> as "weights"!

$$\mathbf{c}_i = \sum_j \alpha_{ij}\, \mathbf{h}_j^e$$

# Dot-product attention: illustrated

In **dot-product attention**, the dot product between every pair of words is used to build a custom, context-dependent vector.

In theory, we can do this at each layer of a neural network.

But the dot product is still a pretty coarse measure of attention.

Can we do better?

# Lecture plan

- “Attention”: high-level introduction and motivation.
- The transformer architecture—is attention all you need?
- The advent of “pre-trained LLMs”.

"RNN + Attention—but throw out the RNN!"

# Introducing transformers

The **Transformer** is a neural network architecture that uses multi-head self-attention, with no recurrent units.

- Use a **fixed context window**.
- No **recurrent connections**.
- Use **self-attention**.
- Have multiple attention "heads" (**multi-head self-attention**).
- Use **positional embeddings**.

"RNN + Attention—but throw out the RNN!"

# Introducing transformers

The **Transformer** is a neural network architecture that uses multi-head self-attention, with no recurrent units.

- Use a **fixed context window**.
- No **recurrent connections**.
- Use **self-attention**.
- Have multiple attention "heads" (**multi-head self-attention**).
- Use **positional embeddings**.

What do these aspects of a transformer remind you of?

A traditional **feed-forward neural language model**!

(Note: this is why you often hear about the "context window size" of models like ChatGPT, Claude, etc.)

"RNN + Attention—but throw out the RNN!"

# Introducing transformers

The **Transformer** is a neural network architecture that uses multi-head self-attention, with no recurrent units.

- Use a **fixed context window**.
- No **recurrent connections**.
- Use **self-attention**.
- Have multiple attention "heads" (**multi-head self-attention**).
- Use **positional embeddings**.

These are new concepts—let's focus on **self-attention** first.

# Self-attention: match-making for words

In **self-attention**, the relevance of each word to each other is calculated in context and shared, informing the model's predictions.

**Query (Q)**: representation of current word, used to score against all other words in sequence.

**Key (K)**: labels for other words in sequence, which we "match" against in our search.

**Value (V)**: represent the "content" of each word, which are weighed by attention scores.

A robot must obey the orders given it…

https://jalammar.github.io/illustrated-gpt2/#part-1-got-and-language-modeling

# Self-attention: match-making for words

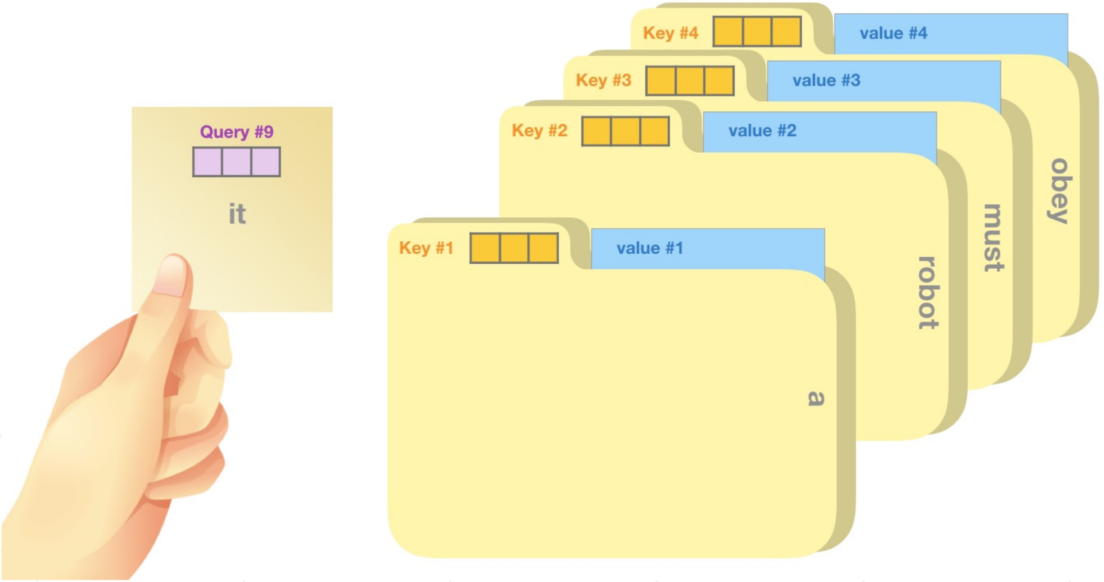
In **self-attention**, the <u>relevance</u> of each word to each other is calculated <u>in context</u> and <u>shared</u>, informing the model's predictions.

**Query (Q)**: representation of current word, used to score against all other words in sequence.

**Key (K)**: labels for other words in sequence, which we "match" against in our search.

**Value (V)**: represent the "content" of each word, which are weighed by attention scores.

A robot must obey the orders given **it**...

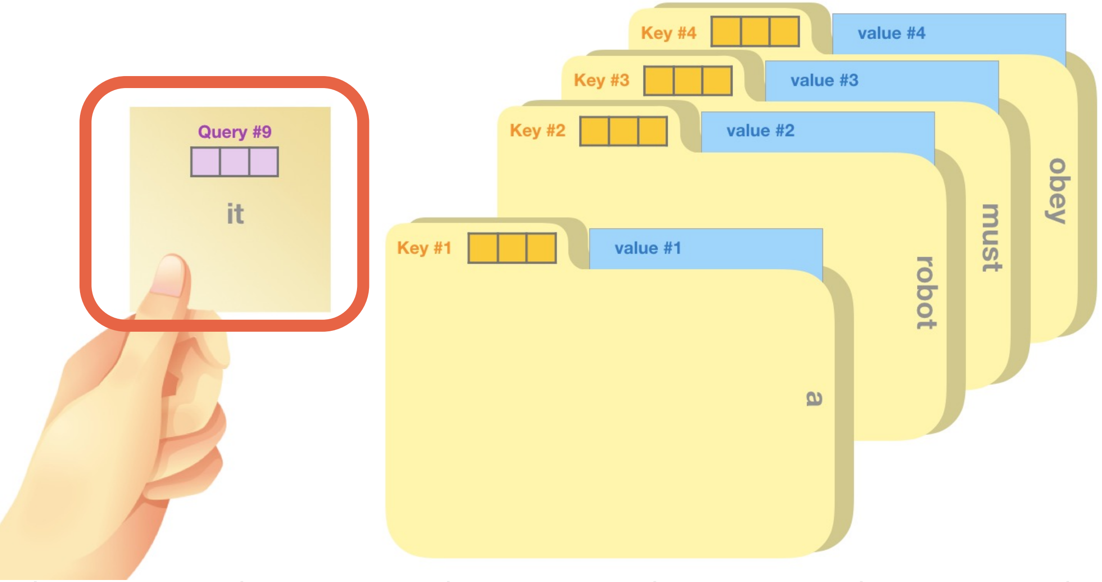


# Self-attention: match-making for words

In **self-attention**, the relevance of each word to each other is calculated in context and shared, informing the model's predictions.

Here, we're looking for words that are relevant to "it".

A robot must obey the orders given **it**...



https://jalammar.github.io/illustrated-gpt2/#part-1-got-and-language-modeling

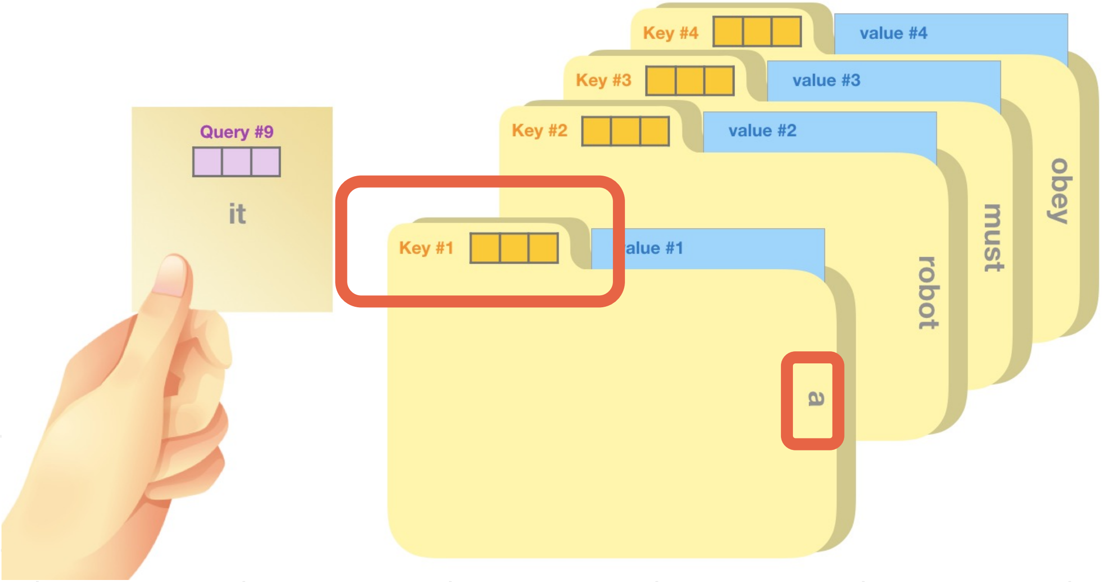
# Self-attention: match-making for words

In **self-attention**, the relevance of each word to each other is calculated in context and shared, informing the model's predictions.

Here, we're looking for words that are relevant to "it".

Key for each word is like a label for "folders" in a filing cabinet.

A robot must obey the orders given **it**…



https://jalammar.github.io/illustrated-gpt2/#part-1-got-and-language-modeling

# Self-attention: match-making for words

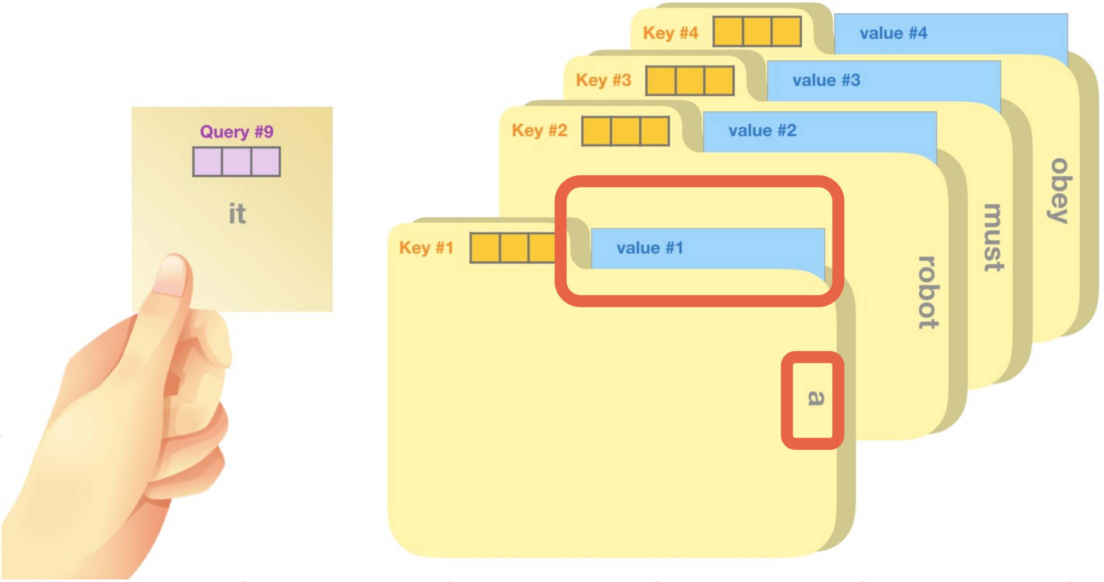
In **self-attention**, the relevance of each word to each other is calculated in context and shared, informing the model's predictions.

Here, we're looking for words that are relevant to "it".

Key for each word is like a label for "folders" in a filing cabinet.

Values are the contents of those filing cabinets.

A robot must obey the orders given **it**…



https://jalammar.github.io/illustrated-gpt2/#part-1-got-and-language-modeling

# Self-attention: match-making for words

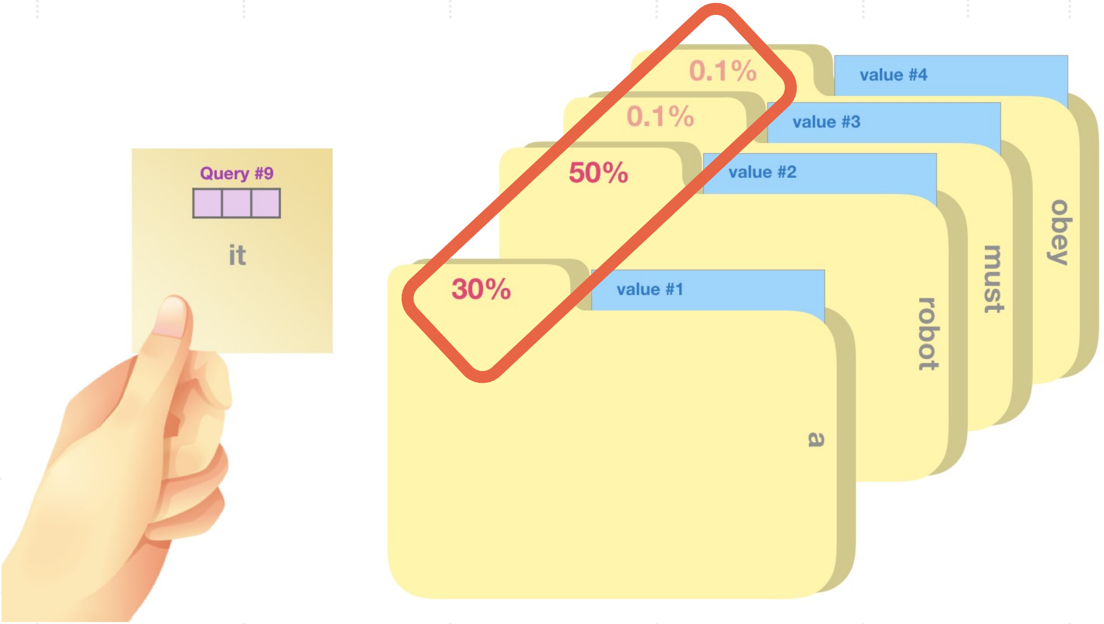
In **self-attention**, the relevance of each word to each other is calculated in context and shared, informing the model's predictions.

To compute **attention score**, multiply query by key vectors for each pair.

$$\text{score}(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{q}_i \cdot \mathbf{k}_j$$

(We then **normalize** and **soft-max** these scores to get a probability distribution.)

A robot must obey the orders given **it**...



# Self-attention: match-making for words

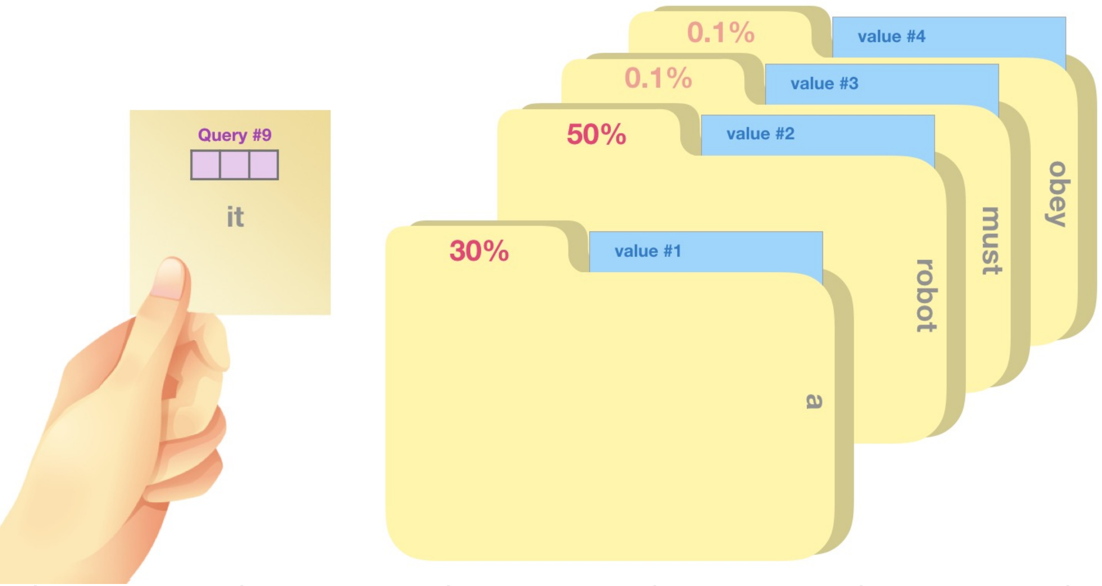
In **self-attention**, the relevance of each word to each other is calculated in context and shared, informing the model's predictions.

To compute **attention score**, multiply query by key vectors for each pair.

Now, multiply (and sum) attention scores by value vectors.

$$\mathbf{y}_i = \sum_{j \le i} \alpha_{ij} \mathbf{v}_j$$

A robot must obey the orders given **it**...



# Self-attention: match-making for words

In **self-attention**, the relevance of each word to each other is calculated in context and shared, informing the model's predictions.

To compute **attention score**, multiply query by key vectors for each pair.

Now, multiply (and sum) attention scores by value vectors.

$$\mathbf{y}_i = \sum_{j \leq i} \alpha_{ij} \mathbf{v}_j$$

A robot must obey the orders given **it**…

| Word | Value vector | Score | Value X Score |
|---|---|---|---|
| <s> | | 0.001 | |
| a | | 0.3 | |
| robot | | 0.5 | |
| must | | 0.002 | |
| obey | | 0.001 | |
| the | | 0.0003 | |
| orders | | 0.005 | |
| given | | 0.002 | |
| | | 0.19 | |
| | | Sum: | |

This is our new **contextualized embedding** for "it".

https://jalammar.github.io/illustrated-gpt2/#part-1-got-and-language-modeling

# Self-attention: match-making for words

In **self-attention**, the relevance of each word to each other is calculated in context and shared, informing the model's predictions.

We compute **attention scores** between each word $w_t$ and every word that comes before it.

In an **auto-regressive model**, we prevent attention from "looking ahead" at future words.

N

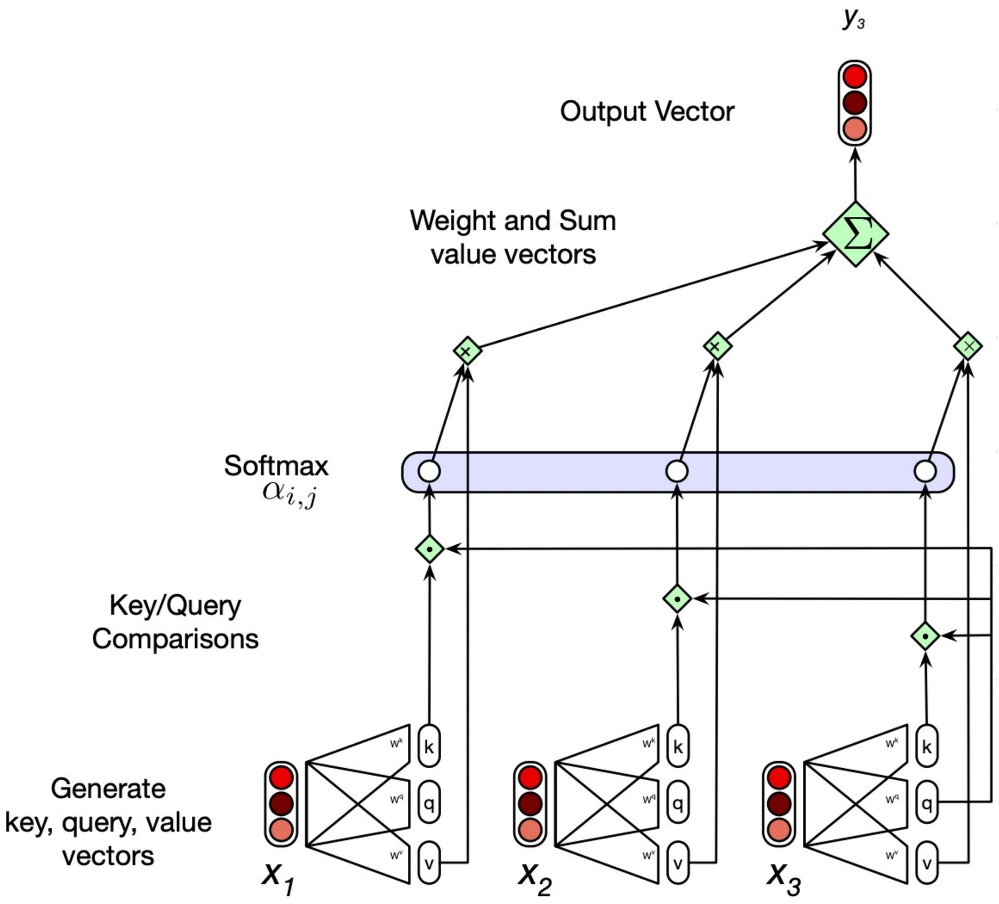
| | | | | |
|---|---|---|---|---|
| q1•k1 | −∞ | −∞ | −∞ | −∞ |
| q2•k1 | q2•k2 | −∞ | −∞ | −∞ |
| q3•k1 | q3•k2 | q3•k3 | −∞ | −∞ |
| q4•k1 | q4•k2 | q4•k3 | q4•k4 | −∞ |
| q5•k1 | q5•k2 | q5•k3 | q5•k4 | q5•k5 |

N

In terms of compute time, how "efficient" is this process?

It's **quadratic**—we must compute dot product between every pair of tokens in the input.

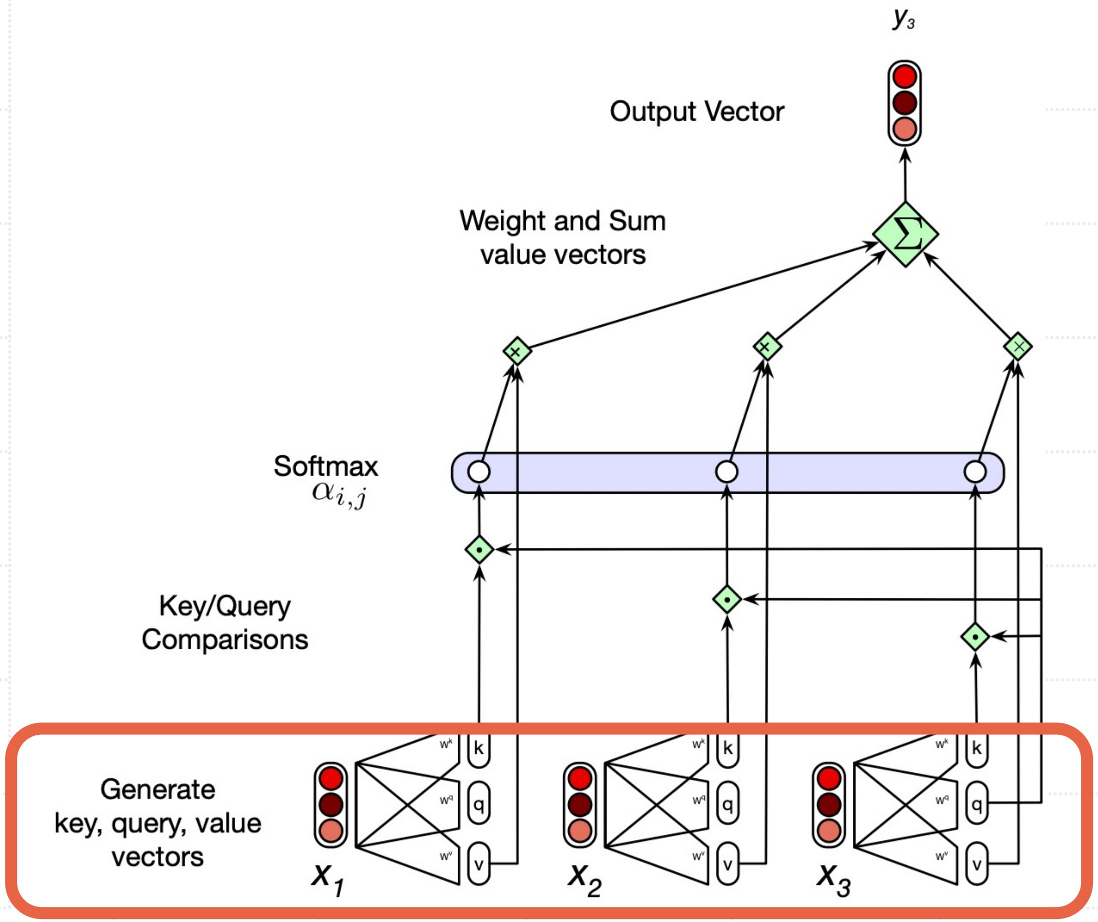
# Self-attention: a closer look




Suppose we are computing **self-attention** for $X_3$.

https://jalammar.github.io/illustrated-gpt2/#part-1-got-and-language-modeling

# Self-attention: a closer look

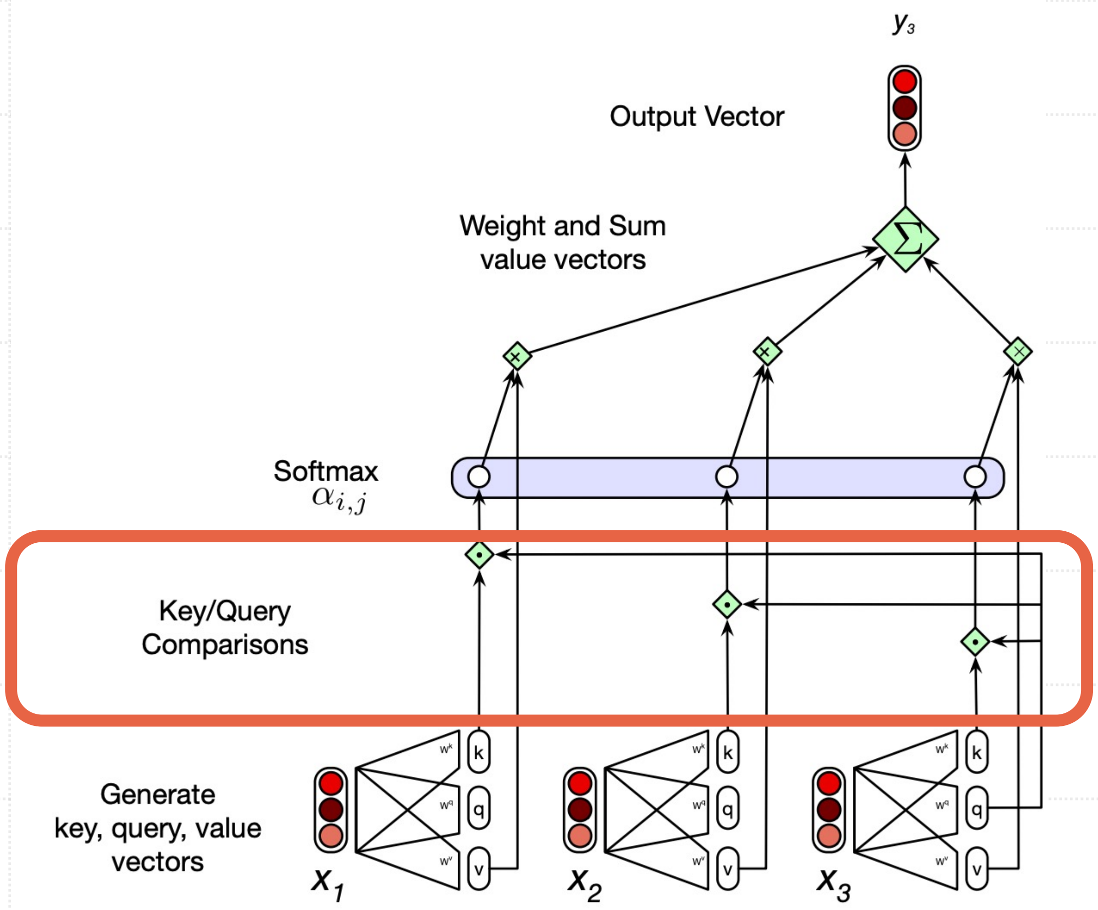



Suppose we are computing **self-attention** for $X_3$.

For each word in sequence, compute **key**, **query**, and **value** vectors.

https://jalammar.github.io/illustrated-gpt2/#part-1-got-and-language-modeling

# Self-attention: a closer look

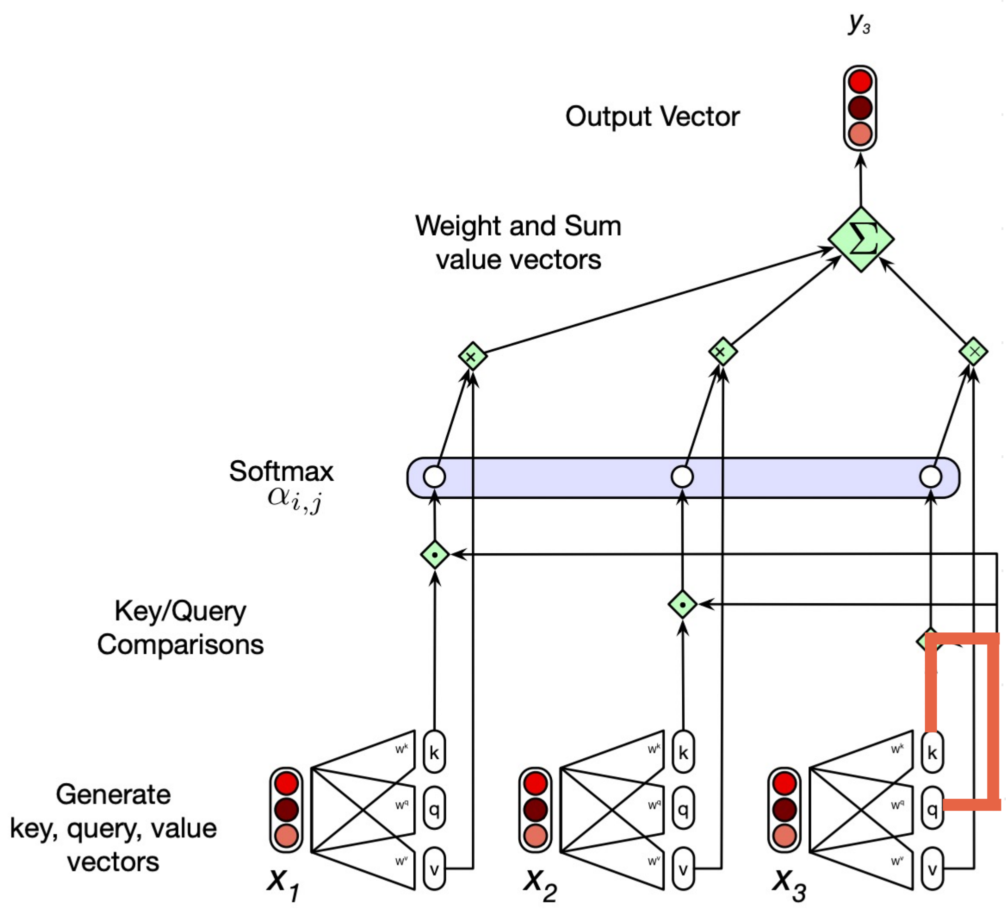



Suppose we are computing **self-attention** for $X_3$.

Compute relevance of $X_1$ and $X_2$ to $X_3$.

https://jalammar.github.io/illustrated-gpt2/#part-1-got-and-language-modeling

# Self-attention: a closer look

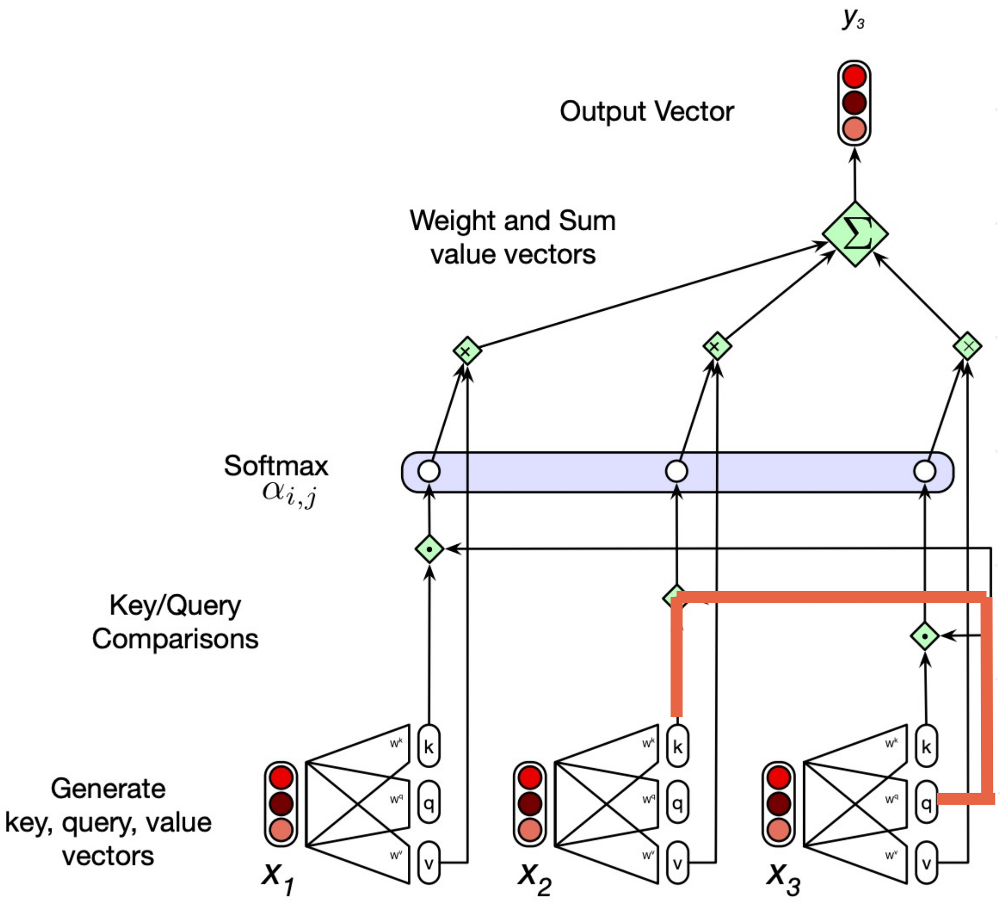



Suppose we are computing **self-attention** for $X_3$.

Compute relevance of $X_1$ and $X_2$ to $X_3$.

https://jalammar.github.io/illustrated-gpt2/#part-1-got-and-language-modeling

# Self-attention: a closer look

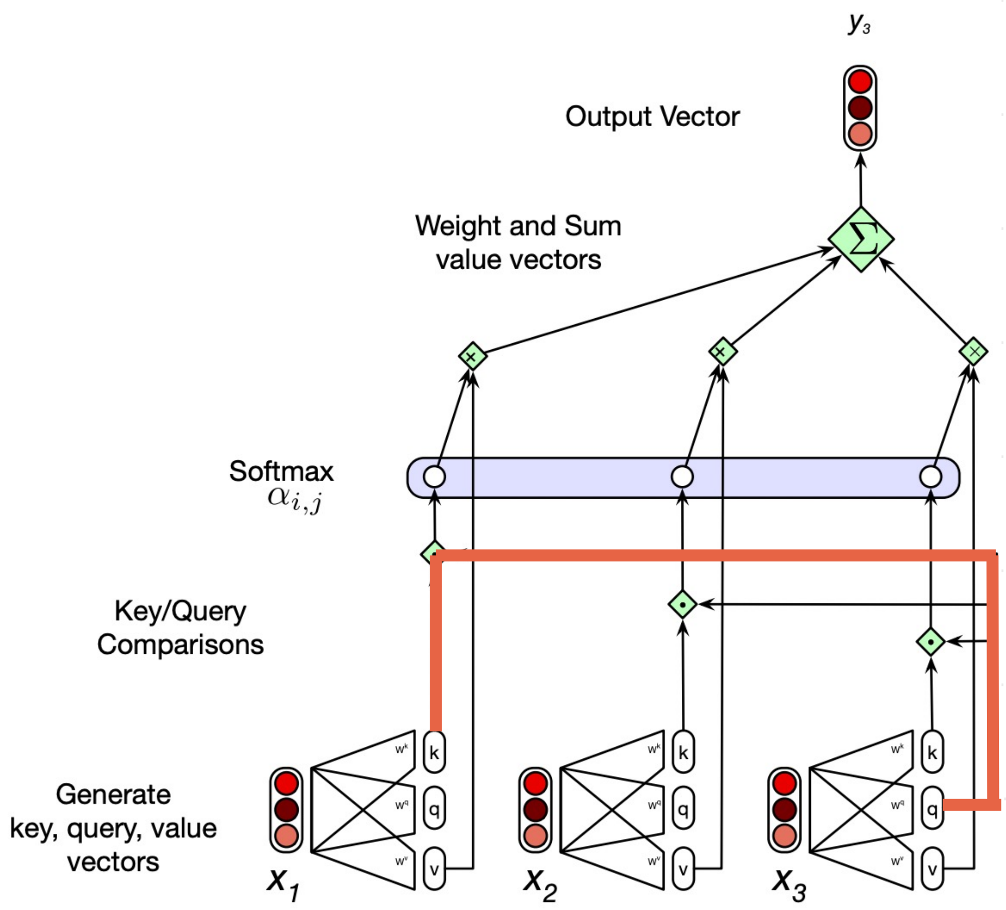



Suppose we are computing **self-attention** for $X_3$.

Compute relevance of $X_1$ and $X_2$ to $X_3$.

https://jalammar.github.io/illustrated-gpt2/#part-1-got-and-language-modeling

# Self-attention: a closer look

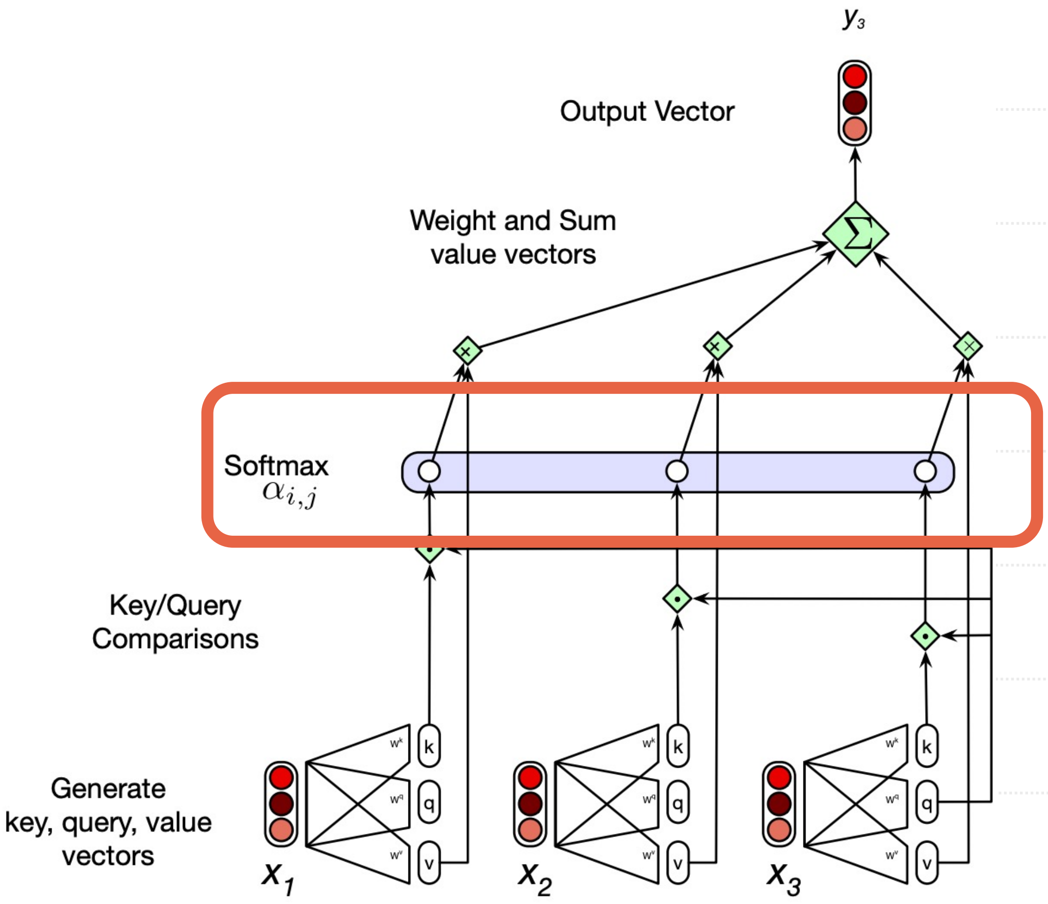



Suppose we are computing **self-attention** for $X_3$.

Compute relevance of $X_1$ and $X_2$ to $X_3$.

https://jalammar.github.io/illustrated-gpt2/#part-1-got-and-language-modeling

# Self-attention: a closer look

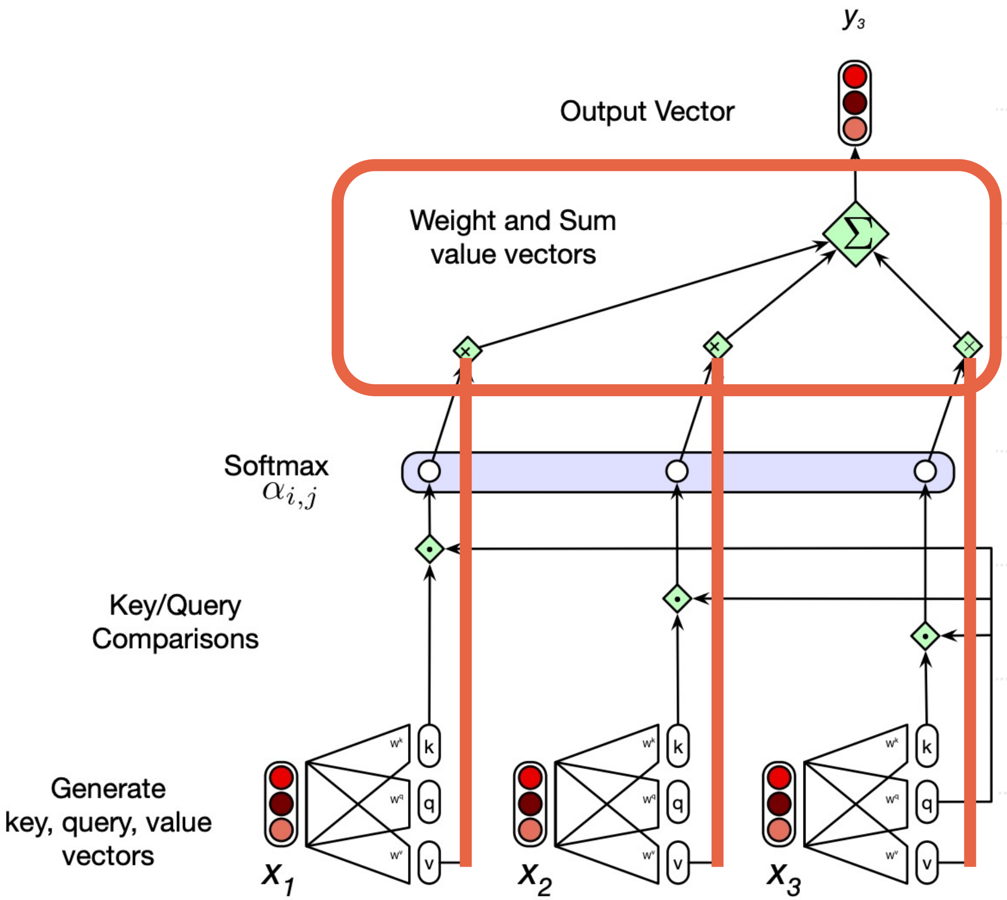



Suppose we are computing **self-attention** for $X_3$.

Soft-max these to get **attention scores**.

https://jalammar.github.io/illustrated-gpt2/#part-1-got-and-language-modeling

# Self-attention: a closer look

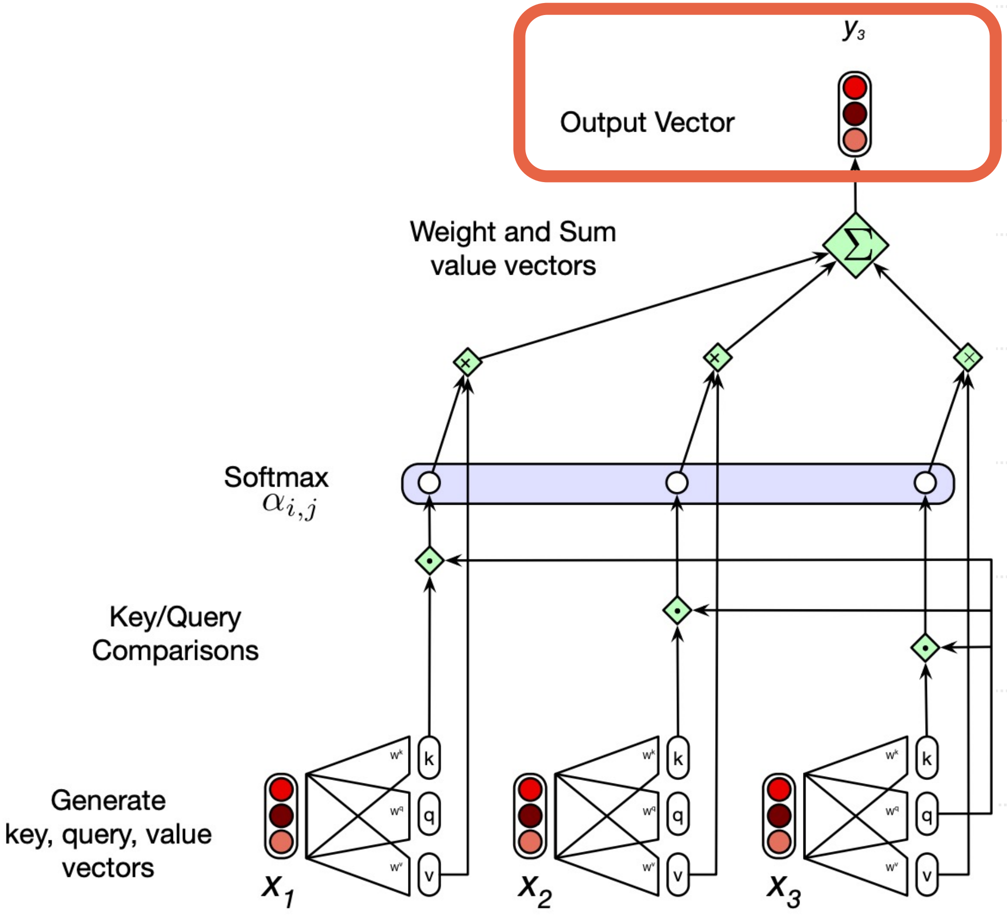



Suppose we are computing **self-attention** for $X_3$.

Use attention scores to weigh the **value vectors**.

https://jalammar.github.io/illustrated-gpt2/#part-1-got-and-language-modeling

# Self-attention: a closer look

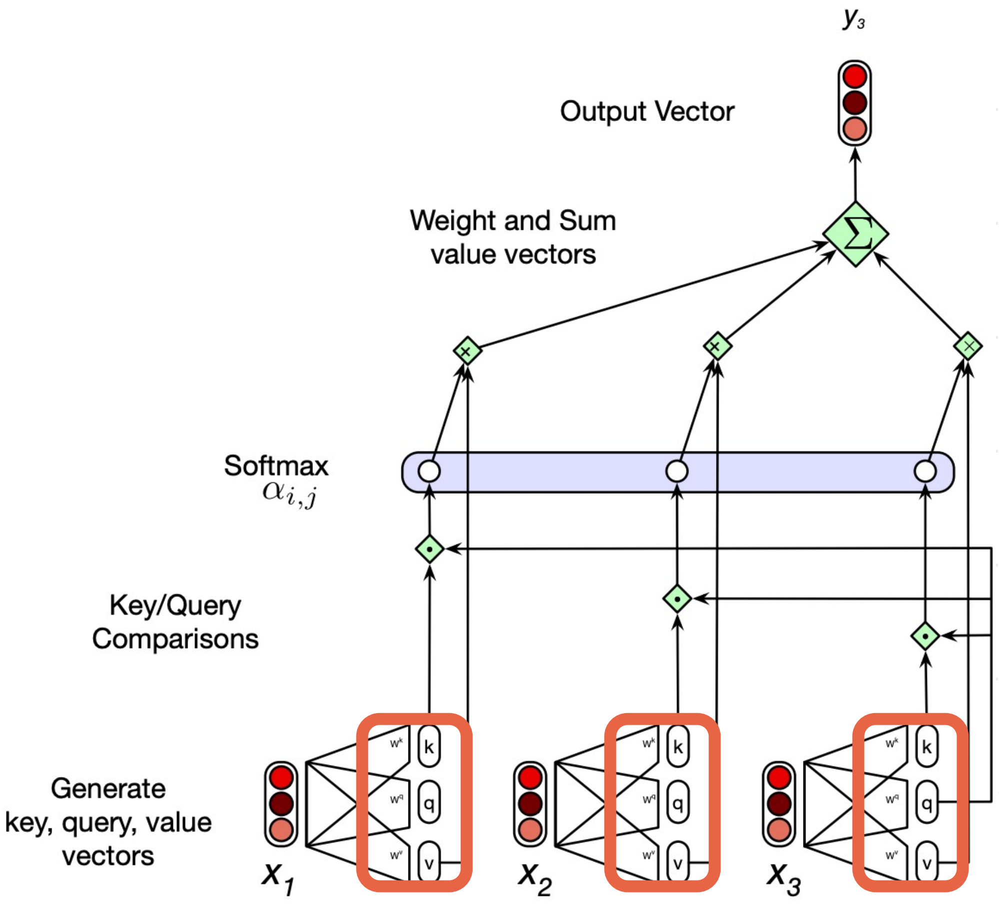



Suppose we are computing **self-attention** for $X_3$.

The result is a new embedding $Y_3$, which "folds in" the relevant information from $X_1$ and $X_2$ into $X_3$.

https://jalammar.github.io/illustrated-gpt2/#part-1-got-and-language-modeling

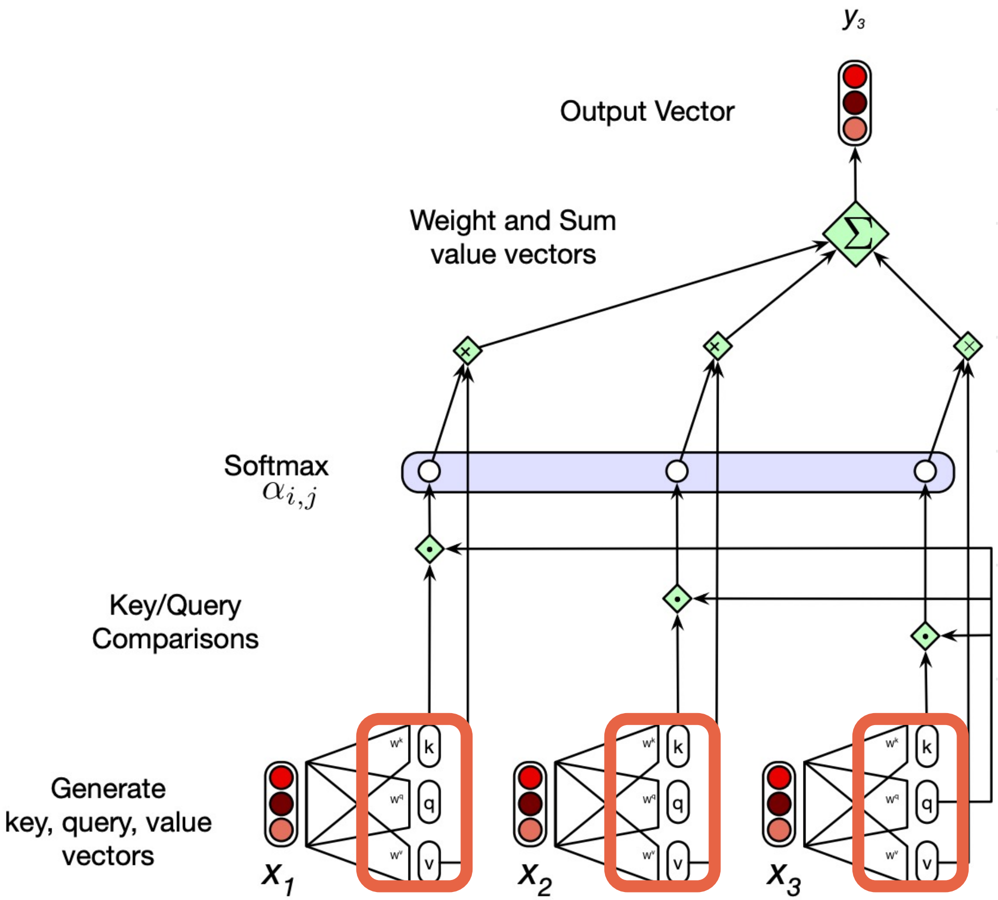
# Self-attention: a closer look




Where do Q, K, V come from?

https://jalammar.github.io/illustrated-gpt2/#part-1-got-and-language-modeling

# Self-attention: a closer look



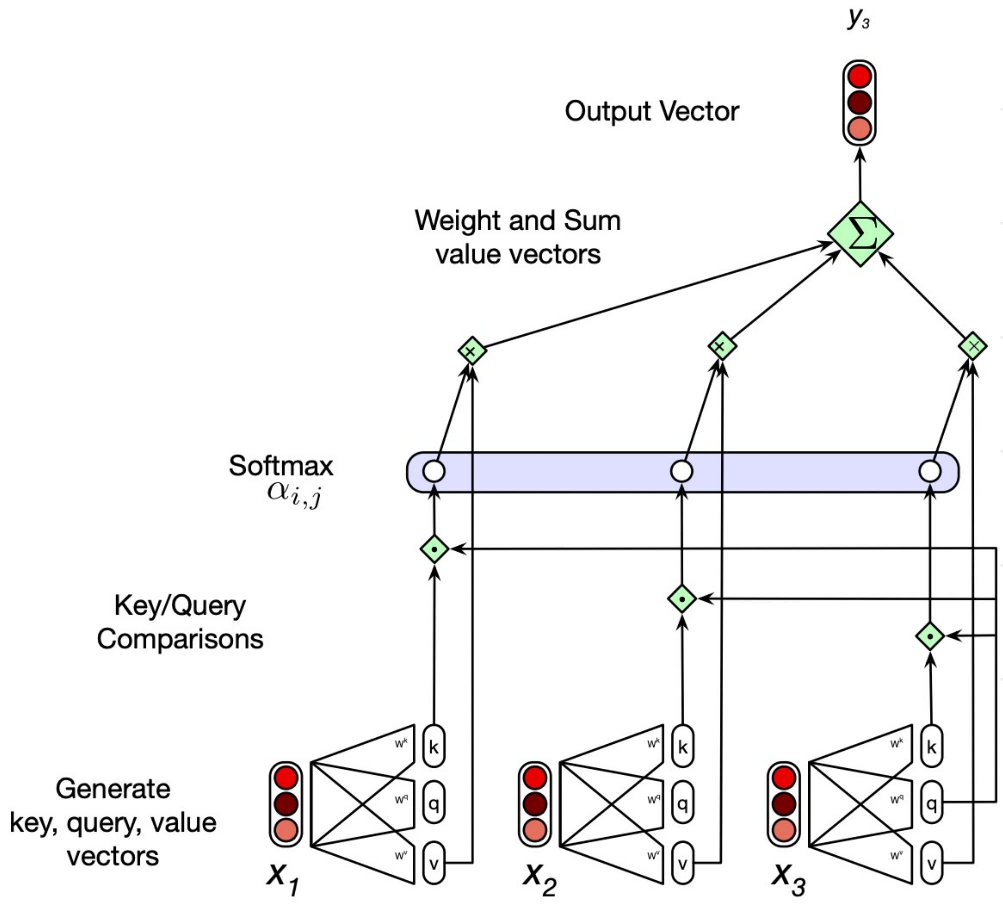

During training, we also **<u>learn weight matrices $W^Q$, $W^K$, and $W^V$,</u>** which we multiply by input **X**.

$$\mathbf{Q} = \mathbf{X}\mathbf{W}^{\mathbf{Q}};\ \ \mathbf{K} = \mathbf{X}\mathbf{W}^{\mathbf{K}};\ \ \mathbf{V} = \mathbf{X}\mathbf{W}^{\mathbf{V}}$$

Learned just like standard weights—by iteratively updating through **back-propagation**.

https://jalammar.github.io/illustrated-gpt2/#part-1-got-and-language-modeling

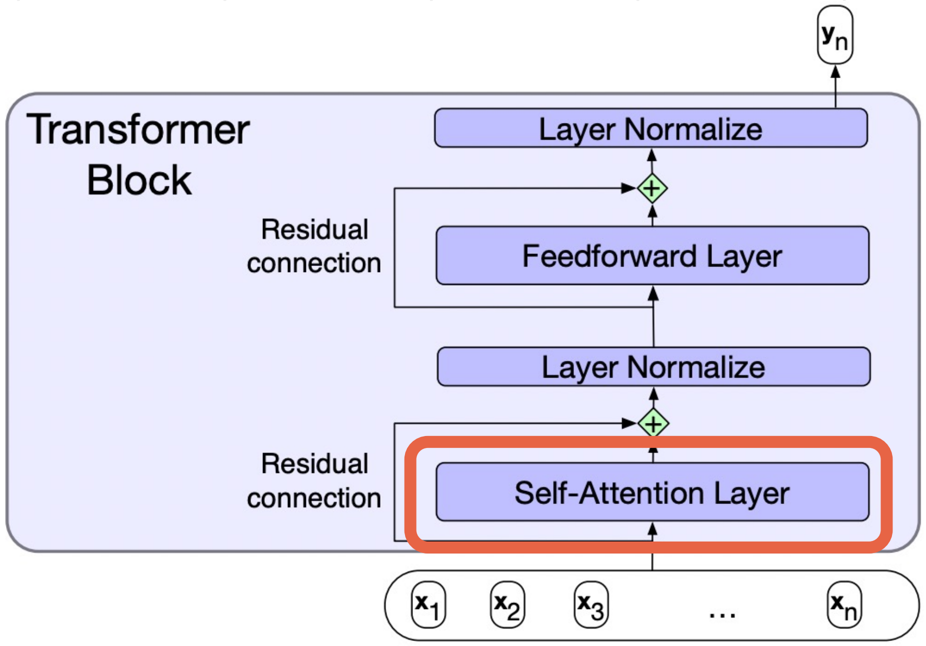
# Self-attention: a closer look




But self-attention is just **one component** of the Transformer…

# The Transformer “block”

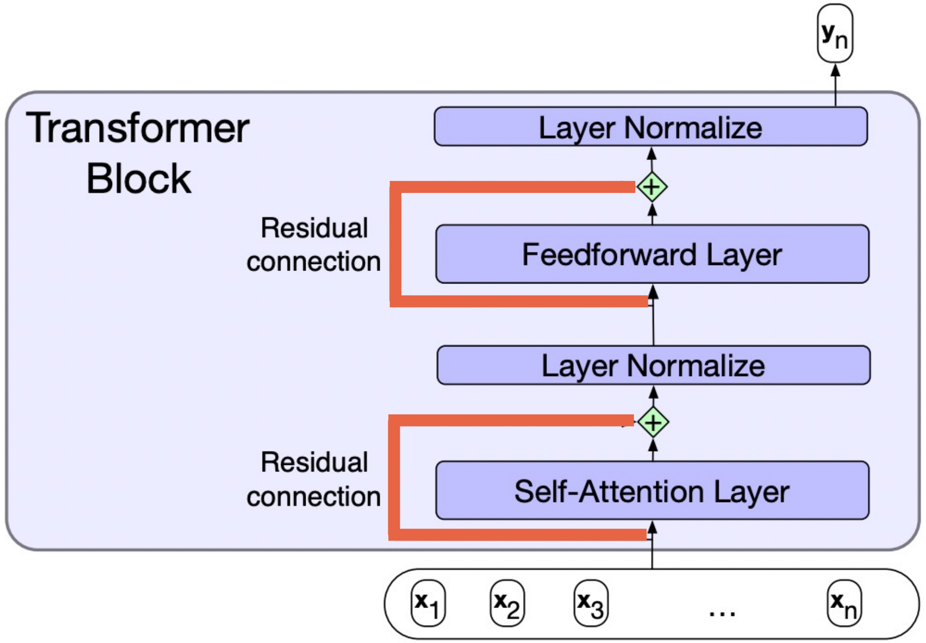
A **Transformer** “block” contains a self-attention layer, feed-forward layers, residual connections, and normalizing layers.




Self-attention: used to compute new, context-dependent representations for each token.

# The Transformer "block"

A **Transformer** "block" contains a self-attention layer, feed-forward layers, residual connections, and normalizing layers.



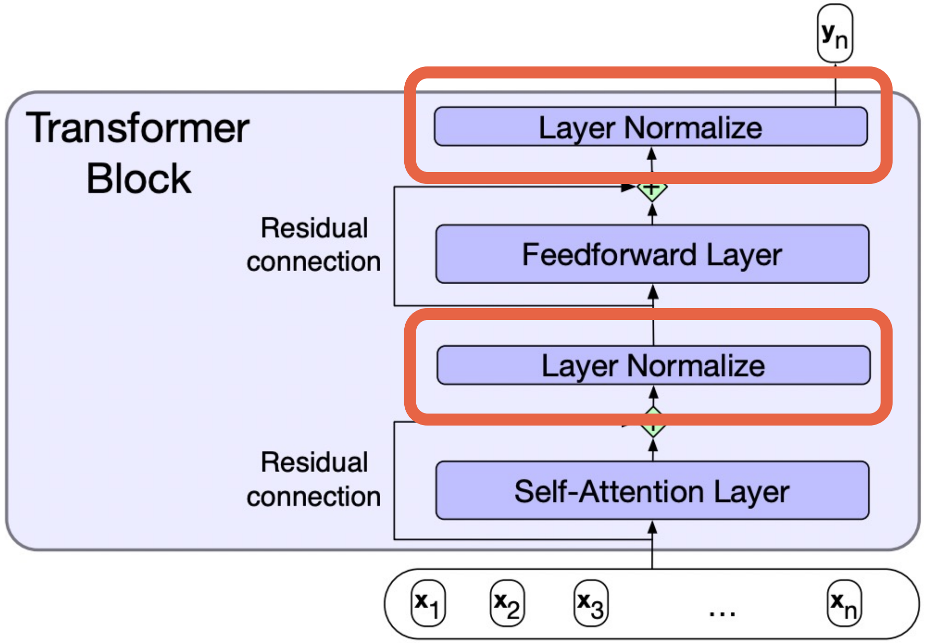

The **"residual connection"** projects directly from a lower layer to a higher layer, without passing through the intermediate layer.

To implement, add a layer's *input* to its *output* before passing it forward.

```
"dog" + Self-Attention("dog")
```

# The Transformer "block"

A **Transformer** "block" contains a self-attention layer, feed-forward layers, residual connections, and normalizing layers.

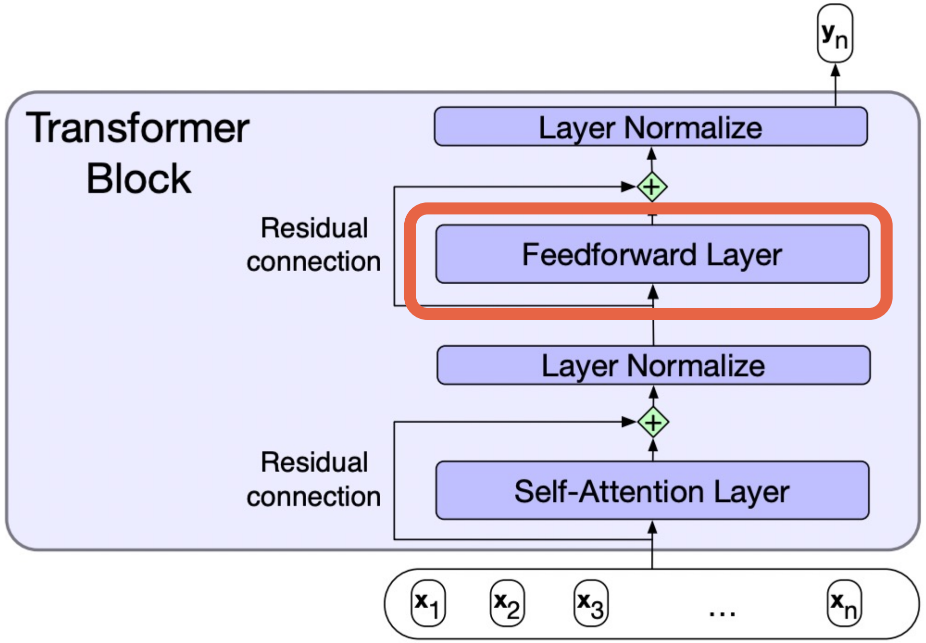



**"Layer normalization"** keeps the values of a hidden layer within a range that facilitates gradient-based training—similar to a *z*-score.

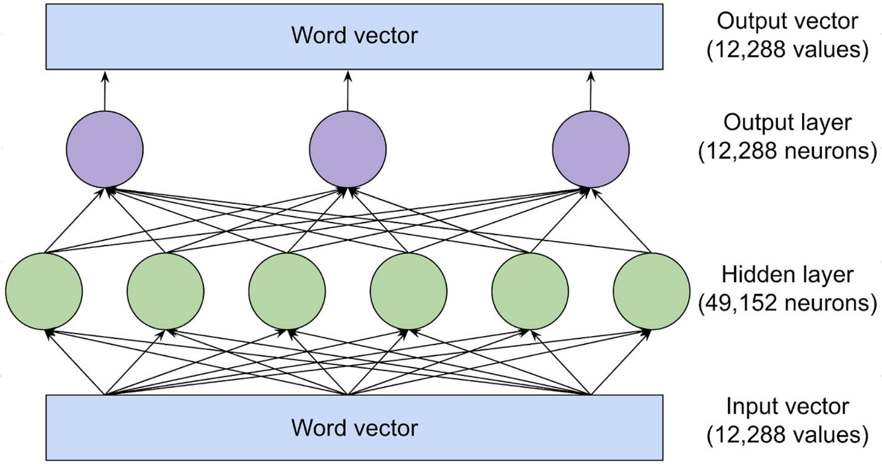
In GPT-2 and GPT-3, this FFN has **two layers**.

# The Transformer “block”

A **Transformer** “block” contains a self-attention layer, feed-forward layers, residual connections, and normalizing layers.



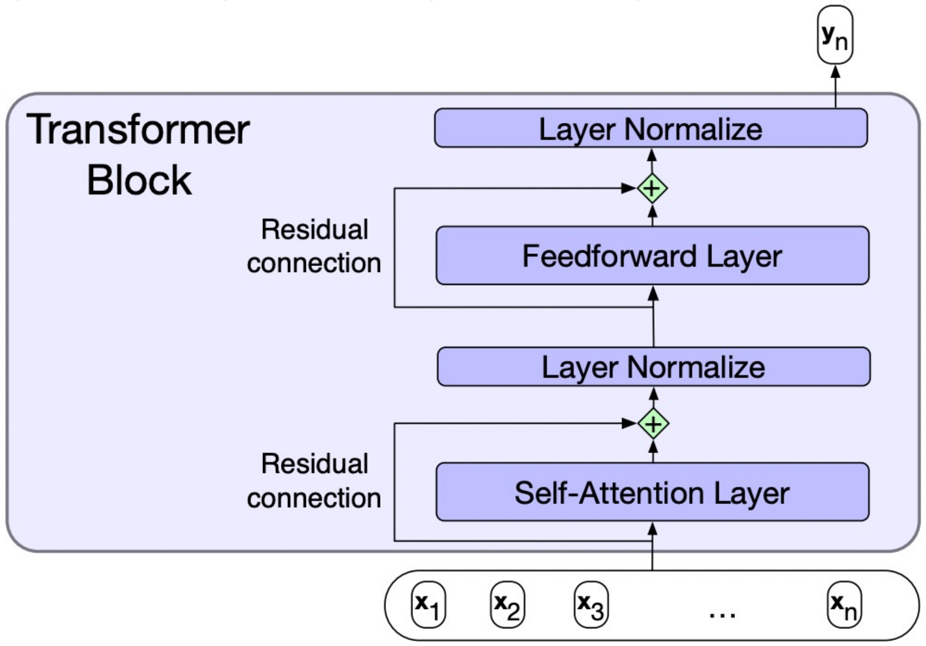

These vectors are then passed to a **feed-forward network**.




Schematic of FFN in GPT-3.

# The Transformer "block"

A **Transformer** "block" contains a self-attention layer, feed-forward layers, residual connections, and normalizing layers.

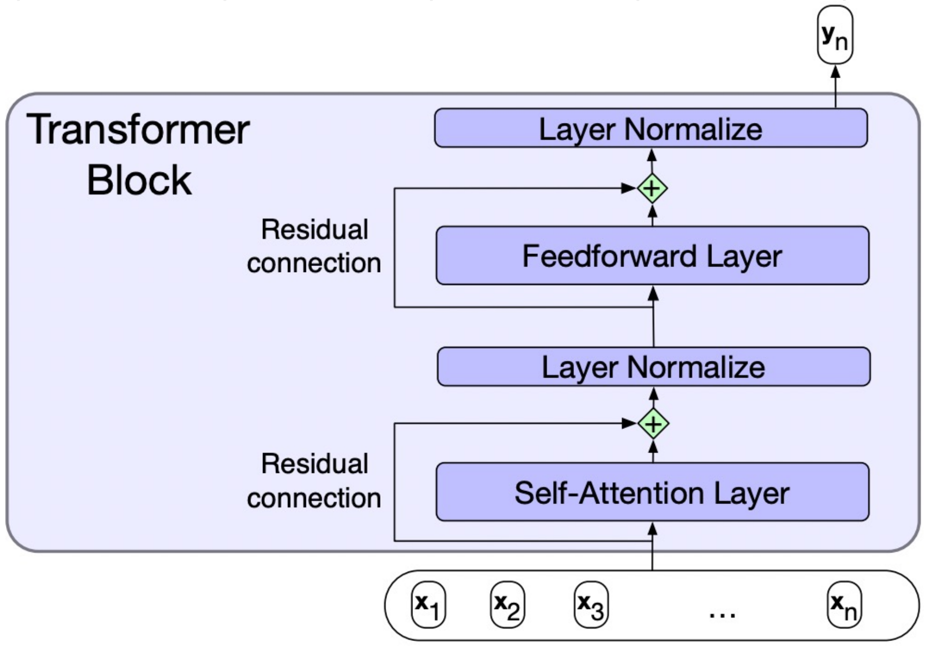


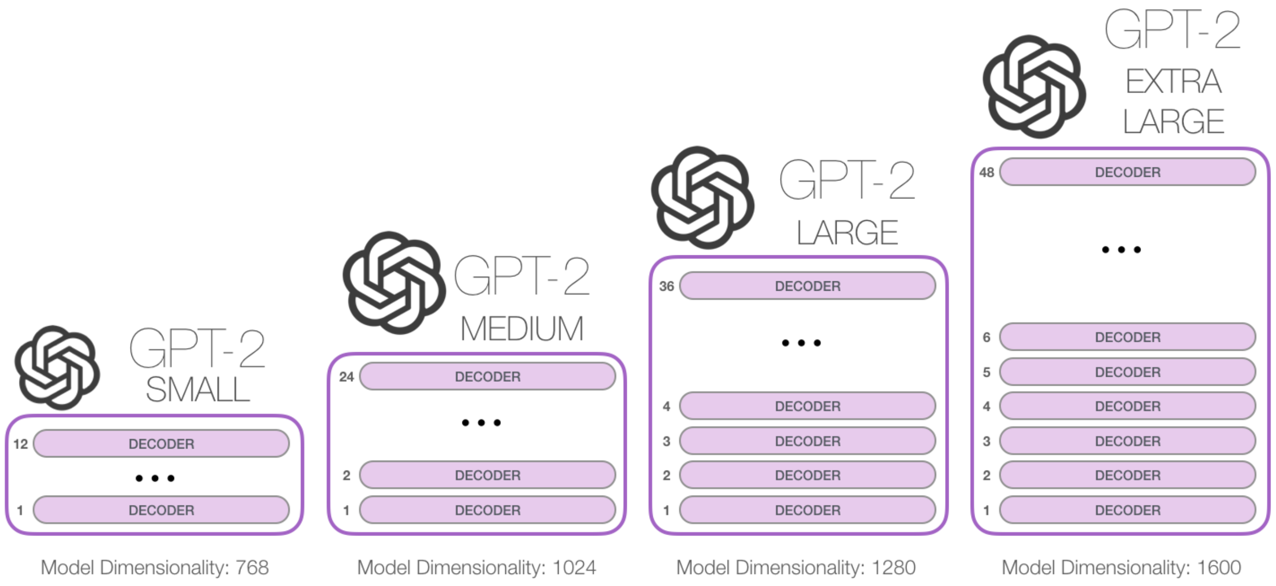

Also called *decoder blocks*.

Models like GPT-2 and GPT-3 have *many* of these!

# The Transformer “block”

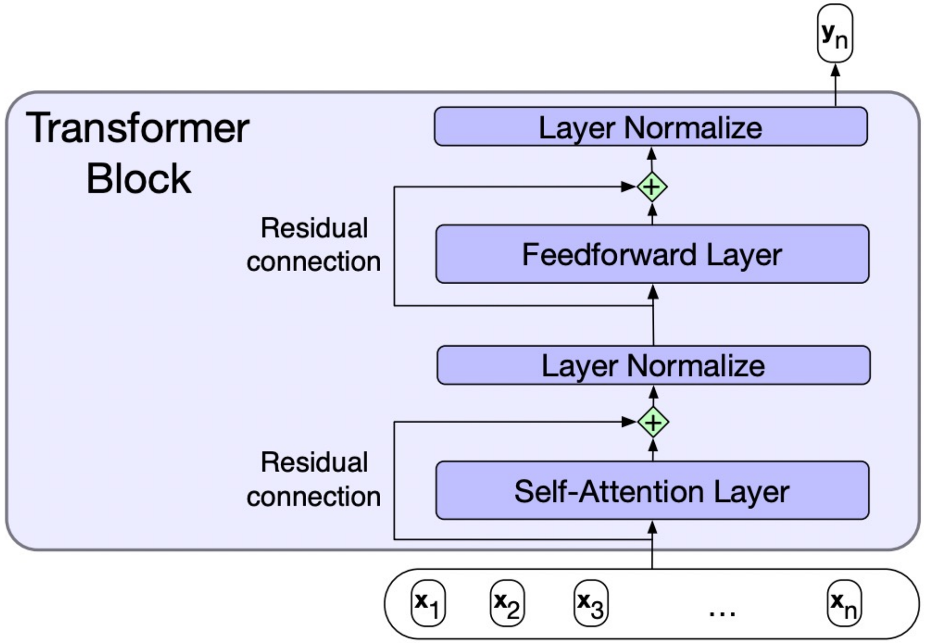
A **Transformer** “block” contains a self-attention layer, feed-forward layers, residual connections, and normalizing layers.



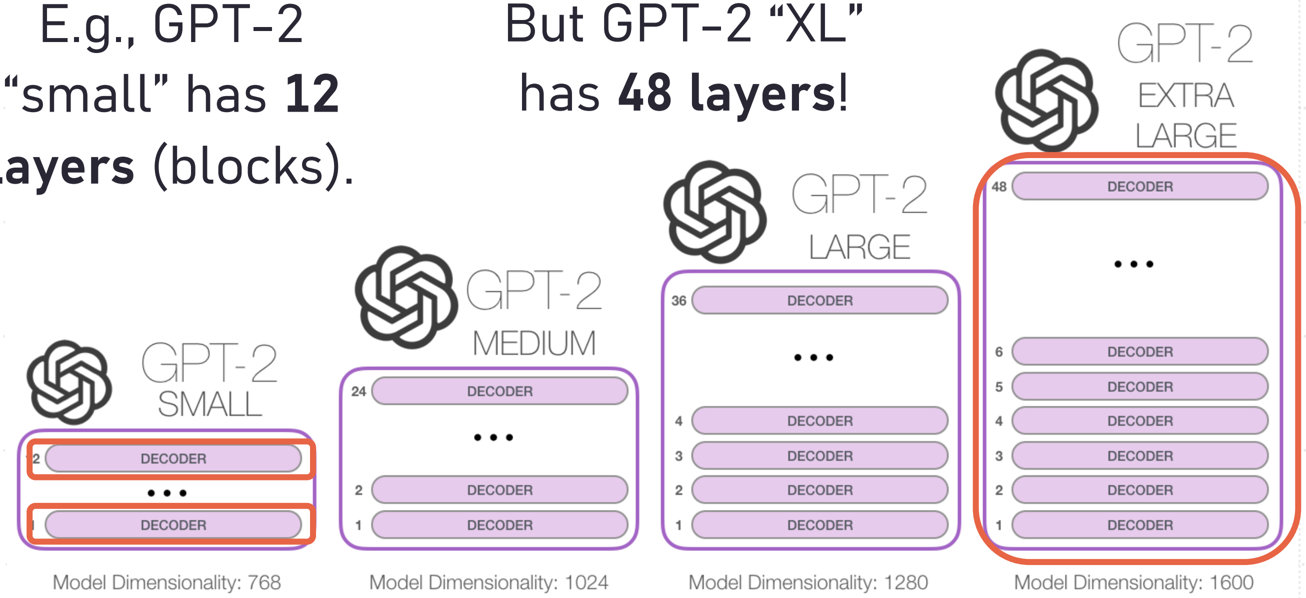
# The Transformer "block"

A **Transformer** "block" contains a self-attention layer, feed-forward layers, residual connections, and normalizing layers.




E.g., GPT-2 "small" has **12 layers** (blocks).

But GPT-2 "XL" has **48 layers**!



"RNN + Attention—but throw out the RNN!"

# Introducing transformers

The **Transformer** is a neural network architecture that uses multi-head self-attention, with no recurrent units.

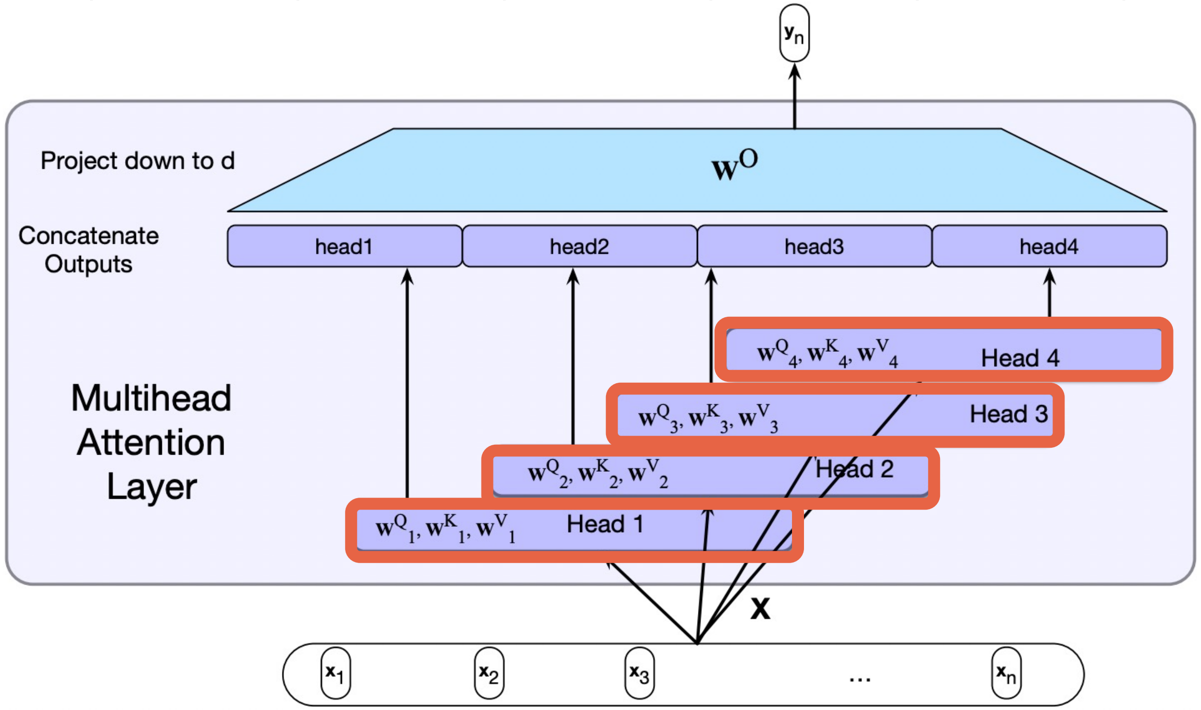
- Use a **fixed context window**.
- No **recurrent connections**.
- Use **self-attention**.
- Have multiple attention "heads" (**multi-head self-attention**).
- Use **positional embeddings**.

We've now covered self-attention— but what's "multi-head" attention?

When we discuss **probing** and **mechanistic interpretability**, we'll talk about research trying to figure out what these heads actually do!

# Multi-head attention

In **multi-head attention**, each layer has multiple attention "heads", each with their own set of learnable weights for producing queries, keys, and values.




Each "head" might learn to track different kinds of relationships.

Over-simplified example:

- Maybe one head tracks syntax.
- Another head tracks proper names.
- Another head tracks events…

"RNN + Attention—but throw out the RNN!"

# Introducing transformers

The **Transformer** is a neural network architecture that uses multi-head self-attention, with no recurrent units.

- Use a **fixed context window**.
- No **recurrent connections**.
- Use **self-attention**.
- Have multiple attention "heads" (**multi-head self-attention**).
- Use **positional embeddings**.

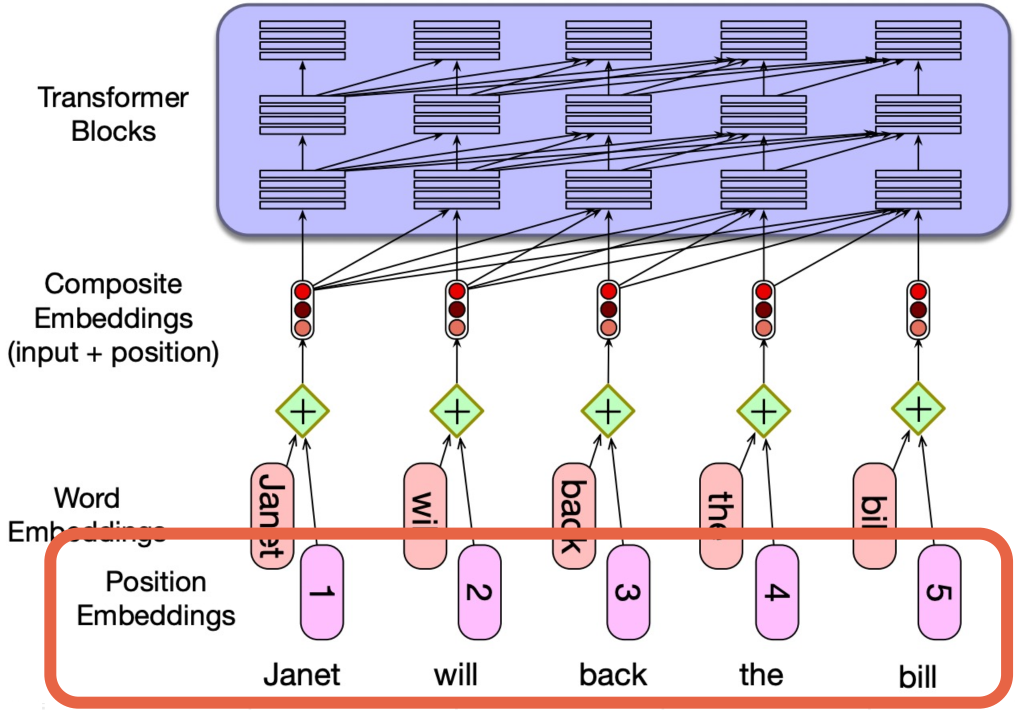
Okay, but what about the **order** of tokens?

With RNNs, order is built into the structure of the network.

Transformers use **positional embeddings** to track order.

# Positional embeddings track order

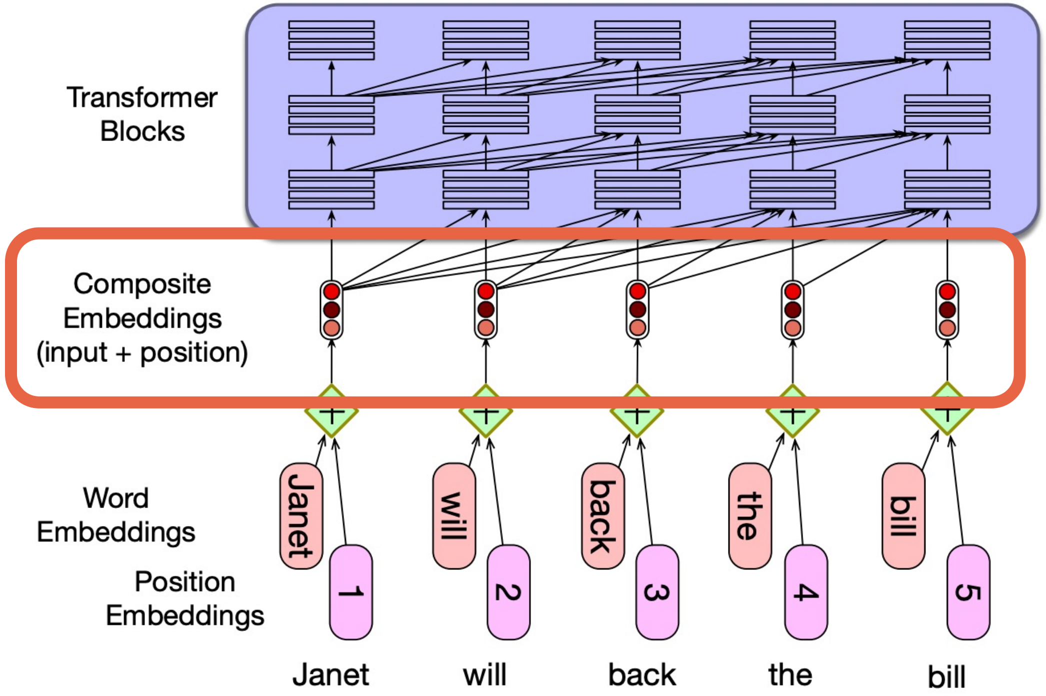
To represent order, input embeddings are combined with **positional embeddings** specific to each position in a sequence.




To learn, begin with random embeddings representing each "position" in a sequence (1, 2, 3, ...)

# Positional embeddings track order

To represent order, input embeddings are combined with **positional embeddings** specific to each position in a sequence.




To learn, begin with random embeddings representing each "position" in a sequence (1, 2, 3, …)

Once learned, we add positional embeddings with word embeddings.

Now, composite embeddings reflect both *word* and its *position*.

“RNN + Attention—but throw out the RNN!”

# Introducing transformers

The **Transformer** is a neural network architecture that uses multi-head self-attention, with no recurrent units.

- Use a **fixed context window**.
- No **recurrent connections**.
- Use **self-attention**.
- Have multiple attention “heads” (**multi-head self-attention**).
- Use **positional embeddings**.

These are very complicated systems! Still lots to learn about why this architecture works.

One practical benefit is (so far) transformers are easier to train than RNNs.

"RNN + Attention—but throw out the RNN!"

# Introducing transformers

The **Transformer** is a neural network architecture that uses multi-head self-attention, with no recurrent units.

- Use a **fixed context window**.
- No **recurrent connections**.
- Use **self-attention**.
- Have multiple attention "heads" (**multi-head self-attention**).
- Use **positional embeddings**.

Under the hood, ChatGPT uses a **transformer** model (plus some other stuff).

"RNN + Attention—but throw out the RNN!"

# Introducing transformers

The **Transformer** is a neural network architecture that uses multi-head self-attention, with no recurrent units.

- Use a **fixed context window**.
- No **recurrent connections**.
- Use **self-attention**.
- Have multiple attention "heads" (**multi-head self-attention**).
- Use **positional embeddings**.

Under the hood, Chat**GPT** uses a **transformer** model (plus some other stuff).

**GPT** = **G**enerative **P**re-trained **T**ransformer

So what's that "pre-trained" word mean…?

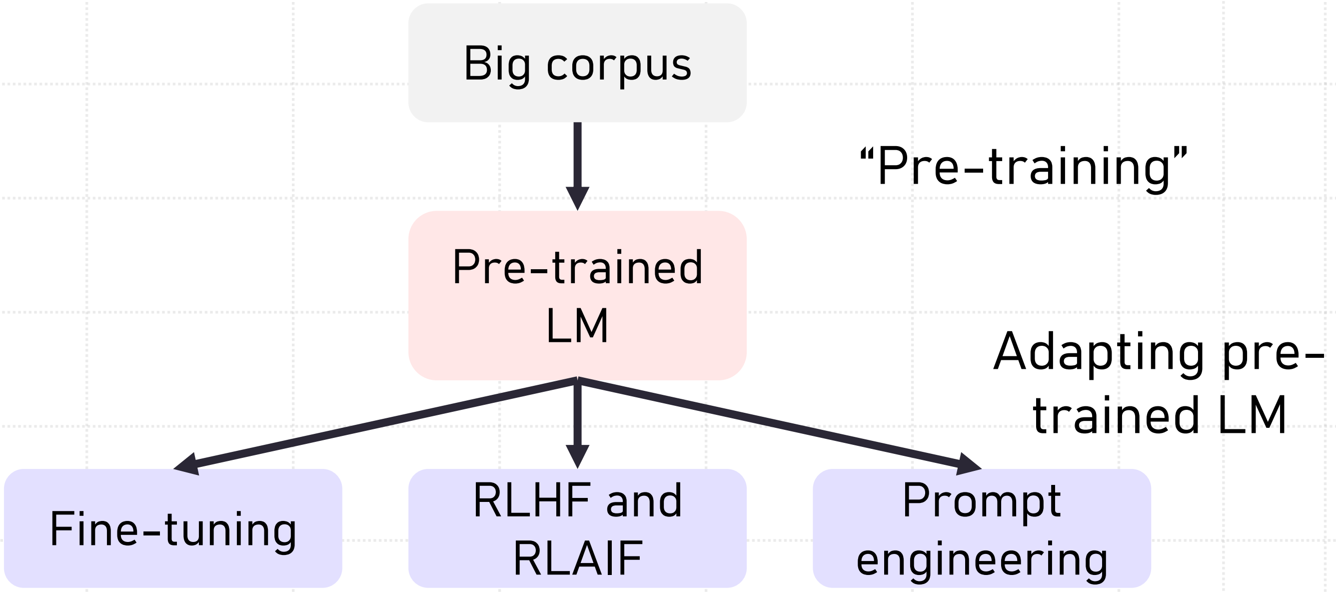
# Lecture plan

- “Attention”: high-level introduction and motivation.
- The transformer architecture—is attention all you need?
- The advent of “pre-trained LLMs”.

# Pre-trained language models

A **pre-trained language model** is a (large) language model that's already been trained on a large corpus using self-supervision.

- "Pre-training" just means training **without a specific end goal in mind** (besides word prediction).
- A "pre-trained" LM can then be **adapted** for specific purposes.
- Practically, it's helpful so we **don't have to train from scratch**!

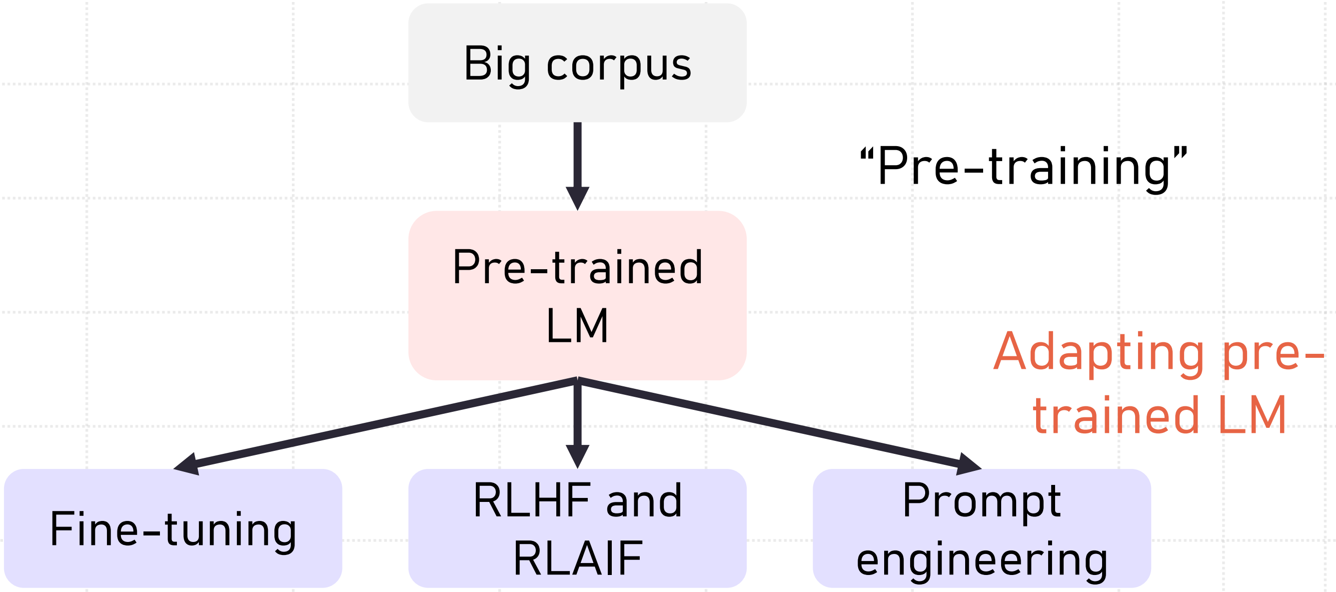


This is what we'll talk about next time!

# Pre-trained language models

A **pre-trained language model** is a (<u>large</u>) language model that's already been <u>trained</u> on a large corpus using self-supervision.

- "Pre-training" just means training **without a specific end goal in mind** (besides word prediction).
- A "pre-trained" LM can then be **adapted** for specific purposes.
- Practically, it's helpful so we **don't have to train from scratch**!



# Summary

- **Self-attention** is a mechanism that allows each word to "look for" other words that are relevant in the input.
- This process creates new **context-dependent vectors** that share relevant information across the words in the input.
- Self-attention a key part part of the **"transformer block"**, which also has other features like a feed-forward network.
- So far, transformers tend to work better than other models like RNNs, and are **easier and faster to train**.
- "**Pre-training**" involves training a model (like a transformer) on a large corpus to learn the "basics" of how language works.